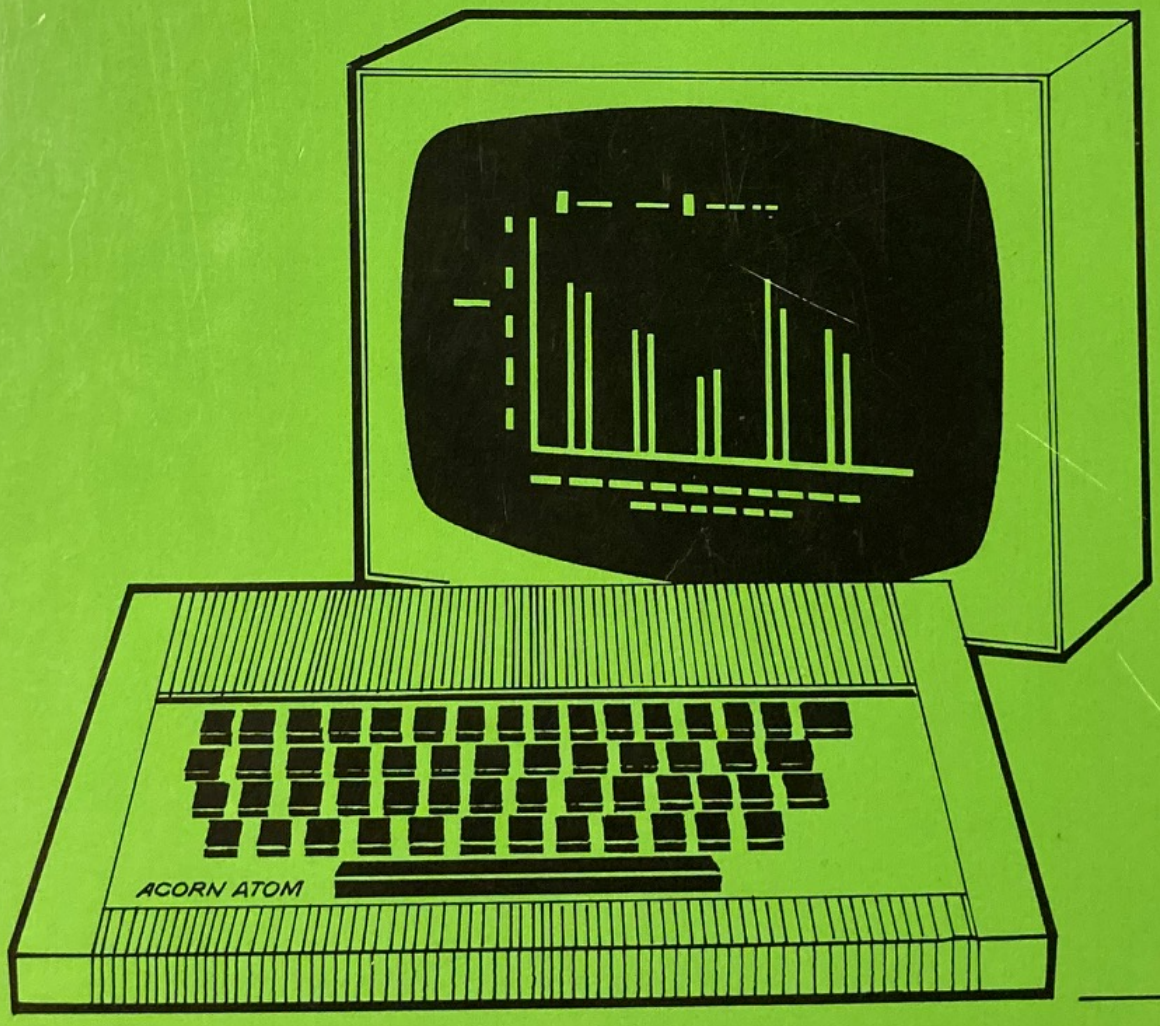
Getting Acquainted With Your ACORN ATOM



by Trevor Sharples & Tim Hartnell

ERROR CODES

2	Too many GOSUBs
6	SUM Checksum error
18	Too many DO statements
29	Unknown or missing function
30	Array too large in DIM statement
31	RETURN without GOSUB
39	Attempt to use variable in LIST
48	COM? Command error
69	Illegal FDIM statement
76	Assembler label error
91	No hexadecimal number after hash
94	Unknown command
	Missing statement terminator or END
95	Floating-point item missing or malformed
109	Number too large
111	Missing variable in FOR
	Too many FOR statements
118	NAME Name error
123	Illegal argument to floating-point function
127	Line number not found in GOTO or GOSUB
128	Argument to SIN, COS or TAS too large
129	Division by zero
	Protected RAM in graphics mode
134	Array subscript out of range
135	SYN? Syntax error
149	Floating-point array subscript out of range
152	GOSUB without RETURN
	FOR without NEXT
156	Assembler error: Illegal type
157	Label not found
159	Unmatched quotes in PRINT or INPUT
165	Loading interrupted
169	Floating-point result too large
174	Significant item missing or malformed
191	LOG or power of zero or a negative number
198	UNTIL with no DO
200	Unmatched quotes in string
208	Unrecognised mnemonic in assembler
216	Illegal DIM statement
230	NEXT without matching FOR
238	Argument to EXP too large
248	Not enough room to insert line

Contents

Foreword, John Galt ----------5
First steps
RANDOM NUMBER GENERATOR ----------6
DECISION MAKER, For/Next, Subroutines----------7
Grand Prix Special
Field, $12, keyboard scan----------16
Digital Clock
Input, Do/Until----------20
Review----------22
Games Interlude 1
TOYING WITH THE MUSE----------23
BACK STABBER----------25
MORDECHAI-MIND----------26
TWENTY-ONE, ATOMIC LOGIC
And, Or, P.$12, Wait----------27
Twenty-one, abbreviated listing----------37
Building a Library----------38
Writing Programs----------39
Games Interlude II
NICOMACHUS----------41
LABYRINTH----------42
TURNING THE TABLES----------47
MASTERING THE GRAPHICS
Clear, Plot, 'Unplot', PEEK, POKE, COLOUR----------48
PLOT----------48
BOUNCING BALL----------51
BREAKOUT----------53
OLYMPIC SWIM----------54
PEEKING BREAKOUT----------57
Higher graphic modes----------61
ARTIST----------63
Adding colour, BROKEN GLASS----------64
Table and graph----------66
LIFE----------66
IMPLODERS----------70
INTERGALACTIC GARBAGE COLLECTOR----------75
PURSUIT----------77
PEEK AND POKE
ROM and RAM, using ?----------79
MALACHI----------85
TARTAN ARMY----------90
ARRAYS
DIM, subscripts----------93
BUBBLE SORT----------96
Multi-dimensional arrays----------97
Byte vectors, word vectors----------99
SWAPPO----------101
MASTERING THE STRING FUNCTIONS
DIM, Comparing strings, LEN, CH, extractions-------103
THE ATOM IN EDUCATION
MULTIPLICATION TABLES----------109
SINES----------109
SQUARES----------109
TEMPERATURE CONVERSION----------110
MULTIPLICATION QUIZ----------110
MONTHLY REPAYMENTS ON A LOAN----------111
COMPOUND INTEREST----------111
DATA SORTS----------111

THE ATOM IN EDUCATION (continued)
ARITHMETIC MEAN----112
PLOTTING, oval, sine curves----113
LOGARITHMS, BASE e, BASE 10----113
AREA OF A RECTANGLE----113
VOLUME OF A CUBE----113
PERIMETER OF A TRIANGLE----114
LENGTH OF SIDE, TRIANGLE----114
AREA OF A TRIANGLE----114
CIRCLE, circumference, area, area of sector----114
VOLUME OF A CONE----115
SURFACE AREA OF A TANK----115
SPHERE, SURFACE AREA, VOLUME----115
RANDOM NUMBERS
Random number graph, distribution, probability----116
CHUCK-A-LUCK----119
SEVENTH HEAVEN----121
CRAPS----123
GETTING TO GRIPS WITH ASSEMBLER
CPU, ROM, binary numbers----125
Hexadecimal, Op codes, $21, $6, LINK TOP----127
KEYBOARD SCANS
BASIC version----130
POKE, SCROLL, SHIFT, CTRL----132
MOONLANDER----133
ASSEMBLER KEYBOARD SCANS----135
TRAFFIC JAM----141
DRAUGHTS, AND OTHER BOARD GAMES
Board numbering----143
Atomic board numbering----144
Micro-DRAUGHTS----145
PROGRAM LISTINGS
DRAUGHTS----149
REVERSI----152
STARBRIGHT----154
TANGLEWOOD----154
FRANTIC----155
HOT SAUCE----155
TRIGGER----156
SPACE STATION----157
LIFE EXPECTANCY----159
TOMBSTONES----160
THE FOREVER WAR----162
PERSONAL ACCOUNTS----166
CHEMIN DE FER----167
MERCHANT OF VENICE (MASTER TRADER)----168
HARLEM----172
MOONLANDER----172
LUNAR LANDER----173
SUB STRIKE----174
ROCK,SCISSORS, PAPER----174
DODGEM----175
MOULIN----176
PURSUIT----176
COLLECTOR----177
IMPLODERS----177
SWAPPO, MALACHI, TARTAN ARMY----178
REFERENCE SECTION, abbreviations, POKE codes, addresses--180

FIRST STEPS

It can be an awful moment when you first get your Acorn Atom and look at the program listings in the manual. "I'll never be able to understand it," you think. But, if your experience was like ours when we starting using the Atom, you soon find you're getting to grips with its language. And the more you use it, the more you appreciate the many fine features on the computer.

But it can take you a long, long time. This book is designed to get you moving much more quickly. To give you a host of programs you can run right away if you like (and then come back and find out how they work later), plus enough information on the unique features of the Atom to enable you to understand them and use them in your own programs.

The book is designed, more or less, to be worked through in order. However, because we wanted to keep each section as self-contained as possible, you'll find a little overlap and duplication in some places. The advantage of this is that you can go straight to the section you want, say PEEK AND POKE, and know that all the vital information you want is in that section.

The programs have been chosen not only because they are valuable in their own right, but because they demonstrate specific functions of the Acorn Atom. We believe the best and most pleasant way to learn about computers is to play games and perform other functions, and then to examine the program listings to try and discover why and how the computer is operating in that program, and -- in due course -- to modify and improve the programs so that they become your own.

If you work through this book with your Acorn Atom turned on at the appropriate moments, you should learn enough about programming to write your own programs. The book assumes you would like to collect a number of different programs that work; that you want to know how and why they work; and -- eventually -- that you want to be able to create and develop your own programs. This book explains how the programs were written, developed and made more complex and/or fun to run, from a simpler "core" program. The specific Atom commands and functions used are explained, and by following through the text, entering the games and other programs into your computer and running them, you should learn the fundamentals of Acorn Atom BASIC without even trying.

Before you read the book, we suggest you just glance through it to get an overall picture of the ground it covers. The more BASIC you know, and the more experience you've had with computers, the further you can read into the book without plugging in your Acorn Atom.

Turn on your TV and plug in the ugly black power converter. Our first program looks at one of the most useful aids for games -- the random number generator. To generate a number between one and, say, 10, you just enter

```
J= ABSRND%10+1
```

Try it now. Clear the screen by pressing BREAK, and once you have the words ACORN ATOM in the top left hand corner, type in

```
J=ABSRND%10+1 press RETURN then enter PRINT J
```

Press RETURN again and a number between one and 10 will appear. If you want the Atom to print a whole series of random numbers you need a program, so you don't have to enter all of the above every time. Enter the following:

```
10      PRINT "NUMBER GENERATOR"'
20      J=ABSRND%10+1
30      PRINT J
40      GOTO 20
```

Now type in RUN and press RETURN. The screen will fill with numbers between one and 10, scrolling upwards as new ones appear at the bottom. You stop the program by hitting the ESC (for 'escape') key in the top left hand corner of the keyboard. Stop the numbers, and type in LIST, then press RETURN to get the program listing back on the screen.

There are a number of things we can learn from this program:

```
10      PRINT "NUMBER GENERATOR"'
```

Firstly, each line starts with a line number. You can use any number between 10 and 32767, but you'll find working in multiples of 10 or 20 will give you enough room to add new lines inbetween others if you need to do so later. Line numbers sort themselves automatically into the correct order. When you ran the program, the Atom acted on the lowest numbered line (in this case 10) and obeyed the instruction PRINT and printed what was between the quote marks ("). The single quote mark (') after the double ones following the word GENERATOR tells the Atom to start its next printing on a new line. Pretty simple, basic stuff. So from this line, you've learned the use of line numbers, the command PRINT and the use of a single quote to send the computer to a new line for its next printing. The next line:

```
20      J=ABSRND%10+1
```

This tells the computer to assign the value generated to the *variable* J (we can use any letter from A to Z for a variable). Then, when in the next line you ask the Atom to PRINT J, it prints the number which has been assigned to J. The ABS makes sure the Atom prints only

positive numbers, and the RND%10+1 ensures the numbers generated lie between one and 10. The next line:

```
30      PRINT J
```

This line tells the computer to print out the number which it has assigned to J. Note that the Atom prints out the successive values for J in four neat little columns. The reason for this will be explained in the section entitled GRAND PRIX SPECIAL, and further discussed in TWENTY-ONE: ATOMIC LOGIC. We will not look into it here, but suggest you might like to try changing the line to 30 PRINT J' and see what effect this has. Now, the fourth line of our original program:

```
40      GOTO 20
```

The computer runs through the line numbers in order, carries out the instructions in each line, and then stops when it comes to an END instruction. In this case, when it gets to the end of the program, line 40 tells it to go back to line 20. It does so, then runs through the program again, and again finds the instruction to go back to line 20.

Now, you've learned that a letter, such as J, can be assigned to a *numerical* value. What is known as a 'string' (a letter preceded by a $ sign, such as $B) can be assigned to a word or words (or any combination of letters, symbols and numbers, but we'll stick to words for the time being). Clear the Atom's memory, by entering NEW then pressing RETURN. Check that it has cleared by entering LIST. Use the BREAK key to get the cursor (and the words ACORN ATOM) up in the top left hand corner. Enter the following program:

```
10      DIMA(10)
20      PRINT "NUMBER GENERATOR"'
30      PRINT "WHAT IS YOUR NAME?"
40      INPUT $A
50      PRINT "OK "$A", I AM GOING TO WORK"'
```

```
60      PRINT "OUT SOME RANDOM NUMBERS"'
70      PRINT "PRESS RETURN"'
80      INPUT #B
90      J=ABSRND%10 + 1
100     PRINT J
110     GOTO 80
```

Try running this program. You'll find that line 30 asks your name, and then accepts it in line 40, assigning the string $A to your name. Line 10 (DIMA(10))makes space in the Atom's memory for your name (and in this case allowed for up to 11 letters, one more than the number in brackets. This is explained in some detail in the string handling section in the PEEK and POKE part of the book, but what we've said here is all you need to know at the moment). Next the Atom uses your name in line 50.

The only other thing to note about this program is the lines 70 and 80 which tell you to press RETURN to continue. The string in line 80 (#B) is just a way (one of a few different ways, as you'll learn in due course) to tell the computer to stop and wait for RETURN to be pressed. This feature is a very useful one in games and we'll be using it time and time again.

You've probably thought by now that it would be good if the listing disappeared once the program was underway. You can do this, and assign a graphics mode (don't worry about that at all at this stage, we'll come to it later) by adding the line:

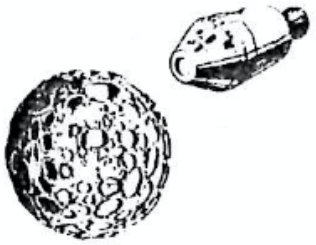
```
5       PRINT #12
```

Add this, by typing it in, the pressing RETURN. It will automatically be assigned its correct position in the listing, which you can check by typing in LIST then pressing RETURN. Once you've run the program a few times with line 5 in place, add the following:

```
45      PRINT #12
77      LINK #FFE3
```

The PRINT &12, you will find, cleared the screen of everything that preceded it on the program, but did not (as will become important later) 'forget' what had been assigned to the string $A. That is, the Atom still remembers your name ($A), so long as you don't turn off the computer, press BREAK, or type in NEW then RETURN. (If you do accidentally press BREAK, typing in OLD, then RETURN, then LIST will get the program back for you.)

Now, rather than just run the program until you get tired of it, and decide to press ESC, you can let the Atom count how many timpes it goes around the loop, that is, how many times it executes the lines 90, 100 and 110. Add the following:



```
76      K = 0
85      K = K + 1
95      IF K > 30 END
```

Now, when you run it you should find the program prints 31 numbers between 1 and 10 and then stops. It does this by assigning the value of zero to the variable (K) in line 76, then adds one to it each time round (so the first time it is, in effect, make K equal to zero plus one, the second time make K equal to one plus one, the third time K equals two plus one and so on) until K is bigger than 30 when the program ENDs. The counting and terminating feature is very useful.

Let's get to our first real program:

```
10      PRINT &12
20      P.'   '
30      P."THINK OF A QUESTION AND"'
40      P."PRESS RETURN FOR A DECISION"'
50      P."ON IT"'
60      INPUT $Q
70      J = ABSRND%3 + 1
80      IF J = 1 P. "YES"
```

```
90      IF J = 2 P."NO"
100     IF J = 3 P. "MAYBE"
110     END
```

Note how in this program the abbreviation P. is used instead of PRINT. There are a number of similar abbreviations which you can use, although they tend to make listings fairly incomprehensible. We will be using more and more abbrieviations as the book progresses, and there is a full list in the appendix. However, for now, we'll only use the shortened form of PRINT, because PRINT is used time and time again -- perhaps more than any other command in BASIC.

Run the program a few times. Notice that the last line of it is END. You need this on all programs to stop your Atom giving a little beep and printing out error code 94. Delete line 110 by entering the number 110 then pressing RETURN. Then run the program again.

Put line 110 back in (110 END). Let's dress up the program a bit. There are at least three things you can do to enhance it:

- -Let the Atom ask for, and use, your name
- -Use CLEAR to make the display easier to read
- -Allow you more than one go before the program stops

Try to amend the program (by slipping things in between the numbered lines and/or changing some lines) to make it do these three things. Don't look at the next listing until you've made an attempt to achieve these ends yourself. The following is just one of many ways to modify the program. There are many other ways to achieve similar ends, and it doesn't matter how you've done it, so long as you've managed to reach the three objectives listed above.

```
5       DIMA(10)
6       DIMB(2)
10      PRINT &12
```

```
20      P.' '
30      P."WHAT IS YOUR NAME"     Note:No question mark
32      INPUT $A
34      PRINT $12
35      P."THINK OF A QUESTION, "$A", AND"'
40      P."PRESS RETURN FOR A DECISION"'
50      P."ON IT"'
60      INPUT $Q
70      J=ABSRND%3 + 1
80      IF J = 1 P."YES"'  Note the single quote mark
90      IF J = 2 P. "NO"'
100     IF J = 3 P. "MAYBE"'
110     P."ANOTHER GO, "$A
120     INPUT $B
130     IF $B = "YES" GOTO 34
140     CLEAR 0
150     P."OK, "$A", BYE BYE"
160     END
```

The program (and others in the book) has been printed with many spaces within a line to make it easier to read, but as -- in nearly all cases -- the Acorn ignores the spaces, and they use up memory, you can leave them out. You'll find you can leave out many brackets as well so line 5, for example, is quite acceptable as

```
5DIMA10
```

However, we won't be listing programs like this, as a general rule, because it reduces clarity. Notice that the Atom, like many other computers, generates a question mark when it is waiting for an input, so you can use this question mark in your print statement by *not* ending the line that is the question with a single quote. Then the Atom will print the question mark on the same line as the question, just where you would have put it when entering the PRINT statement. There is, in fact, a far more elegant way to do this, but we'll leave that till a little later in the book. Modify the above program by changing each CLEAR 0 line into a P.$12, and notice how this both clears the screen, and tells the Atom to start printing at the

top of the screen, rather than the bottom.

Note the way the computer is asked to check if $B is equal to "YES" in line 130. If it finds that $B is equal to YES, control goes back to line 34. There is no need to put in a line IF $B = "NO" GOTO 140 because the Atom moves on until it comes to an instruction it can carry out.

There is another refinement we could add to this program. Try and work out how to make the amendment yourself. We won't be telling you. If you decided you wanted the computer to give you two "NO" and two "YES" to every one "MAYBE", how would you go about changing the program? We'll leave it up to you to do that.

FOR/NEXT LOOPS

You may recall a few pages back we used the variable K to count how many times a certain part of the program was executed. You can use a FOR/NEXT loop to do the same thing. Here are two simple programs, The first uses a counter, the second a FOR/NEXT loop. Study them, and enter them intp into your Atom, and you're sure to be able to understand how they work.

```
10      J = 0
20      P."J"J'
30      IF J = 10 END
40      J = J + 1
50      GOTO 20
```

Now here is the same program, using a FOR/NEXT loop.

```
10      FOR J = 1 TO 10
20      P."J"J'
30      NEXT J ; END
```

As you can see this is much shorter. Use abbreviations, and multi-statement lines (using a semi-colon between statements on the same line) and you can compresss it

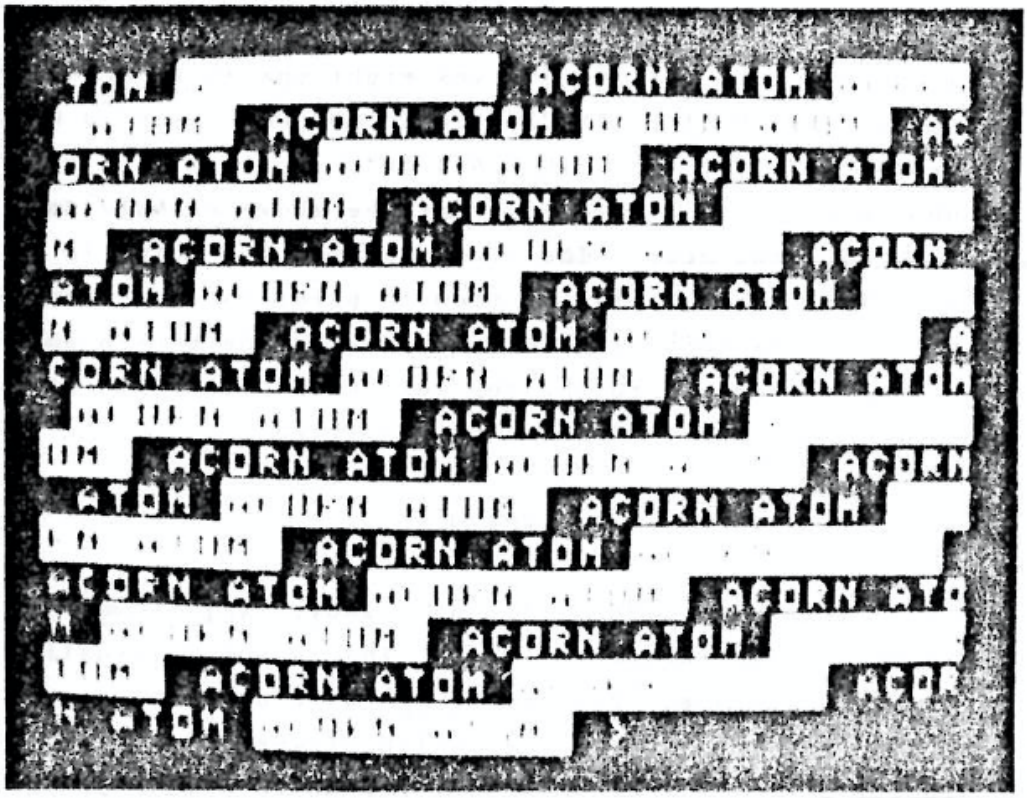
to a tiny program, which does exactly the same as the five line program listed at the start of this section.

```
10F.J=1TO10;P."J"J';N.;END
```

It is not always possible or desirable to use a FOR/NEXT loop as a 'counter' but because it is often a more elegant technique than K = 0, K = K + 1 should always be investigated. You are likely to get into trouble in programs which need you to jump out of loops. That is why, for example, the DRAUGHTS program later in the book does not use loops, but instead sticks to the K = K + 1 idea.

SUBROUTINES

Whenever a computer comes across a GOSUB (GO to SUBroutine) command, it goes to the line specified and follows on until it comes to the command RETURN. The computer then returns to the line *after* the GOSUB command.



GRAND PRIX SPECIAL

This program puts you at the controls of a racing car, on a randomly twisting, turning track. There are a number of useful coding techniques you'll pick up in the course of entering this program, including lines 180 and 190 which give you a routine which scans the kayboard so it knows which key you are pressing (within defined limits) without you having to press RETURN. The program needs at least 2K of text memory, but leaving out the instructions, and paring the PRINT statements should enable you to squeeze it into 1K.

```
10    H=0
20    E=12;X=1;GOS.400
30    @=0;V=0;P.$12;A=0
```

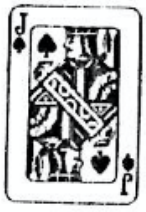
There are a couple of interesting things in line 30. @=0 sets the field width to 0. This means, as you'll see when we look at the development of TWENTY-ONE, the Atom will print out numbers without spaces between them and other print statements. When you switch your Atom on, it sets @ to 8. This reserves eight spaces to print each number, so PRINT "I AM "X" YEARS OLD" where X = 10 would appear as I AM 10 YEARS OLD. Setting @ to 0 tidies up the printing and would print the above example as I AM 10 YEARS OLD. The other interesting thing in line 30 is P.$12. This tells the Atom to print the character 12, which is the control code for clearing the screen and moving the cursor to the top left hand corner of the screen. It is a quick way of clearing the screen.

```
40    F.G=1TO3;R=E+2;  ?#E1=0
```

The instruction ?#E1=0 is very handy. #E1 is where the value for the cursor is stored. The cursor is usually 128, an inverse space. By POKEing #E1 with 0 we change the stored value to 0 (a space) which effectively turns the cursor off. This gets rids of the white blob flashing across the screen when you are printing.

```
50    F.F=1TO15;P.';N.
60    Q=ABSRND%10
```

Note that brackets are not needed in line 60. The Atom can generally work out priorities in a line by itself. The line can be shortened to Q=A.R.%10. and from time to time in this book we will be shortening the line which generates random numbers in this way.



```
70    IFQ > 6 E=E+1
80    IFQ < 4 E=E-1
90    W=E+D+3
100   IF E < 2 E = 2
110   IF E >26 E = 26
120   IF W< 6 W = 6
130   IF W>30 W = 30
```

All the above lines are used to generate the race track, and to keep it within the confines of the screen. Note that after an IF condition you do not have to say THEN (as in some BASICs), as the Atom assumes that a THEN has been implied by the IF.

```
140   E?#81E0=46;W?#81E0=46
```

Line 140 actually puts the track on the screen. #81E0 is the address of the bottom left-hand square of the screen. POKEing this address with 46 will put a full-stop directly into that square on the screen. By placing E before the question mark, the Atom knows to add the value of that variable (that is, the value in this case, of E) to the address to be POKEd (and don't worry if you don't understand POKEing, we get to that a little later in this book). So, as the value of E changes, the address that is to be POKEd with 46 also changes. As the screen scrolls up and the variables change, the race track moves up and across the screen.

```
150   M=ABSRND%2+D
160   IF ABSRND%20<X  M?#81E0=42
```

The preceding two lines decide when and where to put the computer's "crash cars" onto the track.

```
170   P.';V=V+(10-A)*G
180   IF?#B005=191 G.y
190   IF?#B005=127 G.z
```

Now, these were important lines. Address#B005 usually has the value 255 stored in it. However, when the CTRL key is pressed, the value changes to 191; and when SHIFT is pressed, changes to 127. So by checking to see if either of these keys has been been pressed, the Atom can be sent to a particular subroutine to -- in this case -- move your car either left or right. Labels (y and z in this case) are used to speed up the operation. If you GOTO or GOSUB a label (which is an inverse letter obtained by holding down SHIFT before you press the letter, but shown as a lower case letter in Atom publications to make it easy to read) you'll find the Atom gets to the required line more quickly than if you GOTO or GOSUB to a line number. It is also useful if you change line numbers while you are developing a program. Although the use of labels won't save you much time in a short program, they are useful in longer ones.

```
200   GOS.t ;G.60
210y  IF R>0 R=R-1
220   GOS.t ;G.60
230z  IF R<30  R=R+1
240   GOS.t ;G.60
```

The above routines control the movement of your car, making sure you don't drive off the screen.

```
250t  R?#8000=35
```

Line 250 places your car on the screen. It works, of course, on the same principle as line 140. #8000 is the address of the square in the top left hand corner of the screen. R is added to #8000 to put your car

onto the screen.

```
260   IFR?#8020<>32 G.280
```

This line checks if your car has hit anything. #8020 is exactly 30 addresses on from #8000 and so it is the square directly below your car on the screen.

```
270   R.
280   P.&7;R?#8020=163
```

If your car does hit something, the Atom is sent to this line. P.&7 prints out another control code -- a 'beep' this time. The rest of the line turns your car into an 'inverse car' to show you that you've crashed.

```
290   A=A-2*G;IF A>1 A=1
300   P."POINTS = "V*(6-D)';X=X+1
310   F.T=1TO120;WAIT;N.
320   N.;P.&12;P."END OF RACE"'   '   '
330   P."FINAL SCORE = "V*(6-D)'   '   '
340   P."HIGHEST SCORE SO FAR = "
350   K=V*(6-D)
360   IF K > H  H=K
370   P.H'
380   F.F=1TO120;WAIT;N.
390   P.' . '   '   "TO START NEXT RACE PRESS RETURN";
      LINK#FFE3;G.20
```

When the race ends (you get three cars) you get a score, and the highest score so far, printed. Line 390 has the instruction LINK#FFE3. This will make the Atom wait until RETURN (or any other key) is pressed before executing the next command. This is useful when you wish to delay a program for an unspecified time, and you do not need an input.

```
400   P.&12
410   P."GRAND PRIX SPECIAL"'   '   '
420   P."CTRL MOVES YOUR CAR LEFT,"'
430   P."SHIFT MOVES IT RIGHT"'   '   '
```

```
440 IN."DIFFICULTY FACTOR (1-5)"D
450 IF D<1 OR D>5 G.440
460P.''"PRESS RETURN TO START THE RACE";LI.#FFE3;R.
```

This last bit, from line 400, is the subroutine which contains instructions for the race. The 'difficulty factor' (line 440) determines the width of the race track (see line 90). Line 450 is a check to ensure the Atom only accepts a valid input. Once you've mastered the program as given you could introduce such things as oil slicks, 'pedestrians' that wander onto the track, or whatever you can think of it to make it harder, and thus more fun, to play.

DIGITAL CLOCK

In discussing this program, we'll look again at 'turning the cursor off', and P.&12, as well as discussing the use of INPUT and DO/UNTIL.

You'll recall that P.&12 clears the screen and puts the cursor into the top left hand corner of the screen. This means that subsequent printing comes from this point on the screen, rather than along the bottom line. Enter and run the following:

```
  10  P.$12
  20  ?#E1=0
1000  END
```

Run this, and the screen will clear (line 10) and the cursor will vanish (line 20). Hit RETURN a number of times and you'll see the 'greater than' sign move down the left hand side of the screen. The cursor is still there, but invisible.

Now, type in ?#E1=197. The cursor is back, but it is some weird, malformed creature. Enter ?#E1=128, then press RETURN and our old friend, the normal cursor, is back. The invisible cursor is very useful in programs where you want to print, but don't want to suffer a bad case of the flying white blobs.

Enter the next program, which creates an inaccurate digital clock, and after you've got it running on the right time, come back to the book to read the discussion on the program.

```
 10 P.$12
 20 INPUT"HOURS"H,"MINUTES"M,"SECONDS"S
 30 P.$12;?#E1=0
 40 DO
 50 P.H,M,S'$11
 60 S=S+1
 70 IF S=60 M=M+1;S=0
 80 IF M=60 H=H+1;M=0
 90 IF H=13 H=1
100 F.A=1TO52;WAIT;N.;UNTIL0
```

Line 20 shows a very flexible feature of Acorn BASIC. The line actually takes the place of all the following:

```
20  P."HOURS?";INPUT H
21  P."MINUTES?";INPUT M
22  P."SECONDS?";INPUT S
```

Whatever you place in quote marks after the word INPUT will be printed, complete with a question mark, and the computer will automatically start a new line after accepting the input.

The rest of the program is fairly self-explanatory, adding seconds each run through until a minute is clocked up, and every 60 minutes, adding one to the hour figure. Line 30 (DO) and the end of line 90 (UNTIL0) deserve a mention.

DO/UNTIL is a useful function in the Atom's skills, and one which is found in few other microcomputers.

DO/UNTIL is used when a FOR/NEXT loop will not be suitable because the terminating value of the loop is not known, or when you wish a program to carry out a routine over and over again until a specified condition is met. The condition is always placed after the word UNTIL (such as UNTIL $A = "MAGIC" or UNTIL X = 9). Everything between the DO line and the UNTIL is executed over and over again until the end of the world, or the condition is satisfied (whichever comes first).

REVIEW

Before we continue, it may be worth having a quick look over the ground we've covered so far in this book. Ideas we've discussed include:

- *Line numbers (1 to 32767)*
- *Random numbers (A = ABSRND%20)*
- *Use of double (") and single (') quote marks in PRINT statements*
- *The use of variables (A, B etc.) to stand for numbers (or to have numbers assigned to them)*
- *The use of a string ($A) to halt execution of a program*
- *CLEAR 0 and P.$12 to clear the screen (and $12 to move the PRINT position to the top left hand corner of the screen)*
- *The use of a counter (K=K+1) in a GOTO loop*
- *Using an array (DIMA(10)) to accept a string input (or rather, to prepare the Atom to accept a string input)*
- *The use of @ (field)*
- *?#E1=0 to 'turn the cursor off'*
- *POKEing into the screen*
- *A routine to scan the keyboard to see if you are pressing CTRL or SHIFT*
- *The use of labels rather than line numbers for GOTO or GOSUB destinations (as in* IF X=3 GOTO2)

.LINK#FFE3 to halt execution until a key (any key) is pressed

.Use of the INPUT function

.Use of DO/UNTIL

We've come quite a way in a short time haven't we, but there's no time to rest, we must forge onward.

On second thoughts, why not have a rest from all this heavy learning, and take a break with the next few programs.

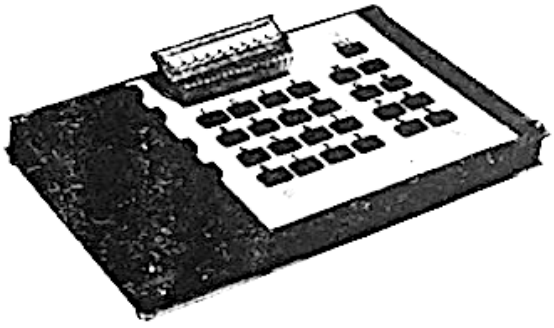
They are designed purely for pleasure, and any learning you do from them will be purely incidental.

TOYING WITH THE MUSE

This is a hard game to play! The program enables the Atom to churn out unlimited quantities of the most excrutiating poetry imaginable (althoygh, to be fair, at times it comes up with quite acceptable poems, certainly as good as some of the 'modern poetry' printed today). Each press of RETURN generates a new verse -- each one allegedly different from the preceding ones. Of course you don't have to stick to our selection of 'seed phrases'.

```
 10 @=0;Q=0;P.$12
 20 P."POETRY"''
 30 P."TO WRITE A POEM PRESS RETURN"''
 40 P."FOR EACH NEW VERSE"
 50 LINK#FFE3;P.$12;G.250
 60 Q=Q+2
 70 IF Q=22 LINK#FFE3;P.$12;Q=0
 80 IF B=0 G.110
 90 IF ABSRND%10>2 G.110
100 B=2
110 IF ABSRND%10>6 G.130
120 P." ";B=B+1;G.140
130 B=0
```

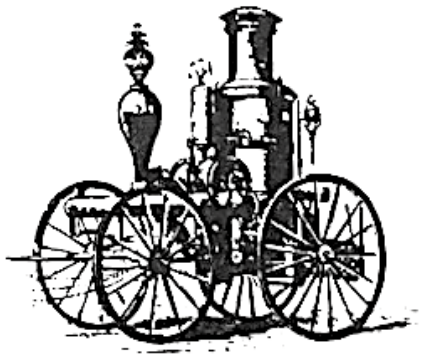
```
140 C=(ABSRND%10/2)+1;D=D+1;E=E+1
150 IF B>0 G.170
160 P." "
170 IF D=1 G.260
180 IF D=2 G.320
190 IF D=3 G.400
200 IF D=4 G.440
210 D=0;P.'
220 IF K>20 G.240
230 G.150
240 P.';B=0;E=0;G.320
250 D=0;E=0;B=0
260 IF C=1 P." SUMMER EYES"
270 IF C=2 P." PHOENIX"
280 IF C=3 P." WINTERED GLORY"
290 IF C=4 P." ANGEL"
300 IF C=5 P." SUN OF ASHES"
310 G.60
320 IF C=1 P." SOFTLY DREAMING"'
330 IF C=2 P." HAUNTED ME"'
340 IF C=3 P." DEEP IN THOUGHT"'
350 IF C=4 P." FLAMING"'
360 IF C=5 P." RISING"'
370 G.60
380 IF C=1 P." AND FROM ASHES"
390 IF C=2 P." NEVER CHANGING"
400 IF C=3 P." AND THEN TURNING"
410 IF C=4 P." SO I WANDER"
420 IF C=5 P." THERE WE TRAVEL"
430 G.60
440 IF C=1 P." FAR AWAY"'
450 IF C=2 P." LONG BECOMING"'
460 IF C=3 P." ...AGAIN"'
470 IF C=4 P." NEVER ENDING"'
480 IF C=5 P." ...ECHOED, ECHOED"
490 G.60
```



BACK STABBER

Even you, with your wide reputation for laziness, could not complain about the thinking you had to do in the last program. However, your brain will have to work mighty hard to keep up with the Atom in this next little game. Full instructions (such as they are) are within the program.

```
 10 CLEAR0;@=0;D=20;G=0;K=50
 20 F.F=1TO20;P."STAND BY,BACKSTABBER"';GOS.1000;N.
 30 P.$12;G=G+1;P."THIS IS ROUND "G''
 80 A=A.R.%10+1
 90 B=A.R.%10+1
100 IF A.(B-A)<2 G.90
110 C=A.R.%10+1
120 IF(C=A)|(C=B)G.110
130 P."MY FIRST NO. IS ";K=2000;GOS.1000;P.A'
140 P.'''

145 P."MY SECOND IS ";GOS.1000;P.B'
150 P.''"YOU HAVE $"D'
160 P.'"HOW MUCH DO YOU BET MY NEXT"'
190 P."NO.LIES BETWEEN "A" AND "B
200 IN.E
210 IFE>D G.200
220 IFE<1 P."COWARD"'
225 K=5000;GOS.1000
290 P.'"MY NO.IS ";GOS.1000;GOS.1000;P.C'
300 P.'';K=50
305 IFE<1 G.20
310 IF(C>A)&(C<B)|(C<A)&(C>B)P."YOU WIN $"2*E'';D=D+2*E
315 GOS.1000;GOS.1000
320 IF(C>A)&(C<B)|(C<A)&(C>B)G.20
340 P."SORRY YOU LOSE $"E/2+1''
350 D=D-(E/2+1)
360 GOS.1000;GOS.1000
370 IFD>0 G.20
```



```
380 P.''"END OF THE GAME"'
390 P.'"YOU ARE BROKE"';P.$7
400 P.'"YOU LASTED "G" ROUNDS"'
410 G.380
1000 F.N=1TO K;N.;R.
```

MORDECHAI-MIND

A similar game to this is marketed under the tradename *MASTERMIND* by Invicta, who bought the rights to the game from an amateur mathematician, Mordechai Meirovich, in 1971. The principle of the game is simple. The Atom creates an array which holds three digits. No two digits within the array are the same, and zero is not used. You input your guess for the number it has chosen as a three-digit number (such as 728), then press RETURN. A correct digit in the correct position in the array will gain you a 'black', a correct element in the wrong position scores you a 'white'. You have 10 guesses to work out what the three digits are. After each guess, the Atom will print the number of the guess, your black score and your white score. Do not use the same digit twice within a single guess - or the poor Atom will get confused.

```
10 DIMAA3,BB3,CC3
20 @=0;P.$12
30 F.A=1TO3;AAA=A.R.%9+1;BBA=AAA;N.
40 IF(AA1=AA2)|(AA1=AA3)|(AA2=AA3)G.30
50 F.E=1TO10;IN.C
60 CC1=C/100;CC2=(C-100*CC1)/10;CC3=(C-100*CC1-10*CC2)
70 W=0;B=0
75 F.D=1TO3
80 IFCCD<>AAD G.95
90 B=B+1;AAD=0
95 N.
97 IFB=3 G.z
100 F.D=1TO3;F.G=1TO3
110 IFAAD<>CCG G.125
120 W=W+1:AAD=0
```

```
125 N.;N.
130 P."GUESS "E"    "B" BLACKS    "W" WHITES"'
140 F.A=1TO3;AAA=BBA;N.
145 N.
150 G.q
160z P."YES,YOU GOT IT IN "E'
170q P."MY CODE WAS "BB1 BB2 BB3
180 E.
```

OK, that's enough playing for now. Back to the hard work.

TWENTY-ONE : ATOMIC LOGIC

We could simply have given you the listing for the game TWENTY-ONE (in which you and the computer each roll a die to get a total close to, or equal to, 21, without exceeding 21, with five rounds to a game) but we thought it far more useful to spell out, step by step, how the program was developed. We suggest you enter each section of the program into your Acorn Atom as you come to it, and read the explanation. You'll learn far more by doing this than you would by just reading through this section and then entering the final program. By the time you've finished this section, you should have some familiarity with the following Atom commands and functions:

.AND

.OR

.PRINT $12

.WAIT

As well as learning these, you should have a little more insight into one way a program can be developed, and this may help you when writing your own programs.

We'll start this program at line 100, which will leave lower numbers for any variables or arrays we discover we need while developing the program.

First up, we need to clear the screen. If we were were going to PLOT graphics, we'd need to specify a graphics mode (such as CLEAR 0, see section starting page 48) but as we're just printing words and numbers a PRINT $12 is all that is needed. So the first line we need is:

```
100 P.$12
```

This will clear the listing from the screen when we press RETURN after typing in RUN. Now, we need to 'roll the die', so we must have a way of getting random numbers from one to six. We'll set up the die (using D as its variable) as a subroutine at 1000, as follows:

```
1000  D=ABSRND%6+1
```

Remember, we're listing the programs with spaces to make them easier to read, but --as you know -- in most cases you do not need to enter the spaces because the Atom can manage quite well without them. Let's check that line 1000 is really capable of producing random numbers in the range we want, from 1 to 6. Add the next line, then run the program and see what happens.

```
1010  P.D;G.1000
```

If all is well, you'll see the cursor flying across the screen, printing out random numbers between one and six. But, as you can see, it does so very, very quickly, far too quickly for you to be even able to read them properly to check they do lie between one and six. We'll add a loop to slow them down. Enter the following:

```
1010  P.D
1020  F.Z=1 TO 1000;N.
1030  G.1000
```

Run this, and you'll see the cursor moving at a much more manageable speed, printing out the numbers in four little columns. Remember, the 'field' is set to eight (we

discussed field in the prelude to GRAND PRIX SPECIAL) unless we tell the Atom otherwise. Change the final number for the loop in line 1020 to 20 (so it reads F,Z=1TO20), then to 2000 to see the effect this has on the speed. Now, we're going to use this little part of the program as a subroutine, we need a RETURN statement. Delete line 1010 (by entering the number, then pressing the RETURN key) and change line 1030 to:

1030 R. *and also add*
990 E.

Every program must include END to stop your computer beeping rudely at you, and giving you error code 94.

To play the game, the human rolls the die as many times as he or she chooses, and then allows the computer to roll. So we need a mechanism to allow the human to tell the Atom a roll is desired, and also a way of showing the accumulated total. It is easy to tell the Atom we wish to roll the die, and add the figure on the die to our score. First we need a variable to 'hold' the player's score. We can't use D or Z because they are already in use. Let us use P for player. We need to set P to zero at the start of each round, so add:

```
110  P=0
```

While we're at it, we may as well give the computer a variable for its score. We'll use C (for computer). Add it to line 110 by adding a semi-colon then C = 0 so the whole line reads:

```
110 P=0:C=0
```

The playing mechanism can be very simple. Add:

```
120 P."ENTER 1 TO ROLL DIE,"'
```

Make sure you add the single quote after the double quotes,

to tell the Atom you want the next PRINT statement to start on a new line. Now add:

```
130   PRINT "OR 2 TO STAND"'
```

Run that, then type in LIST and press RETURN to get your listing back on the screen. A line is needed to accept the "1" or "2", so add:

```
140   INPUT B
```

When there are only two alternatives (in this case 1 or 2) you often need only to specify what the computer should do for one of the alternatives and let the other one take its course. Enter the following and that last, slightly obscure statement, should be clear:

```
150   IF B = 2 G.200
160   GOSUB 1000
170   P=P+D
180   PRINT "YOUR TOTAL IS "P'
190   GOTO 120
```

Run this, entering "1" when prompted for the first few times, then enter a "2". When you enter 1 you'll see the words "YOUR TOTAL IS" come up, followed by a number. When you enter 2, the Aco-n Atom just jumps over the part which rolls your die and adds to your score so -- as we said earlier -- you need only to specify the reaction to one answer to a prompt (in this case, to "2") and let the other one just flow on.

There is one disadvantage of the output as it now stands. You only see your growing total, without getting a display of the number you rolled. So, let us add a line for this.

```
165   P, "YOU ROLLED A "D'
```

Now there is something odd about the output. When you run this, you'll find you get something like:

```
YOU ROLLED A        4
YOUR TOTAL IS       17
```

There will be a vast space between the end of your sentence and the number. Now, remember the example we gave in the introduction to GRAND PRIX SPECIAL where we talked about someone who was 10 years old. The 'vast space' above is because -- as we pointed out -- the Atom has automatically set the field to eight, and has divided the screen into four columns of eight characters each. To change the field, so the numbers immediately follow the words, you need to add:

```
90  a=0
```

You can set the field at any number from zero upwards, but it is pointless to set it any higher than 15, as you'll discover if you set it to 2000. Now that you've added line 90, run the program again. Unless you've left a single space after the words before the close of the quotes you'll now find the number squeezes up tightly against the last letter in each sentence. Put a single space after the words "A" and "IS" in lines 165 and 180.

Now we come to the interesting part, the part of the program where the Atom makes decisions and acts on them. Let's look at the rules of the game. If either the player or the computer gets a total higher than 21, the opponent wins. So the first thing the computer must do is check whether or not the player has exceeded 21. This is fairly simply done by adding the following line, which is the first one the Atom will consider after you've decided to stand, by entering a 2.

```
200  IF P> 21 GOTO 500
```

Having decided that the player has not exceeded 21 (which would give the computer an automatic win in that round),

the Atom must now roll the die. Remember we set (in line 110) C for the computer's score. Now, the computer should roll the die if its score is less than the player's score.

```
210   IF C < P GOSUB 1000
```

Line 210 is known as a IF/THEN statement because it is of the form IF something is true THEN do something (*IF the house is on fire THEN scream*), although in Atom BASIC we do not have to spell out the THEN. The intelligent computer works this out for itself (as we said when discussing lines 70 to 130 in GRAND PRIX SPECIAL). We must now display the computer's result, which we can easily do by writing similar lines to those we used for the player in lines 165, 170 and 180. Do this now for the computer, before reading our version.

```
220   P. "I ROLLED A "D'
230   C=C+D
240   P."MY TOTAL IS "C'
```

That's our version. It doesn't matter how you did it, so long as your version displays the result of the roll of the die (D), adds this to the computer's total (C) and then displays the new total (C+D). The computer needs a way to assess the score. It needs to know if C is less than P. So, change line 210 to the version below, and add the line numbered 250.

```
210   GOSUB1000
250   IF C < P GOTO210
```

Now at this point you have a program which will almost play the game. Run it a few times and see what happens. It is vaguely satisfactory, but it would be better if it displayed your total while it was rolling. So add:

```
215   P. "YOU STAND ON "P'
```

This helps a bit, but you've probably noticed the computer's moves are made far too quickly to keep track of them. We can use the delay loop in the subroutine beginning at line 1000 to slow things down a bit. Add

```
245   GOSUB 1020
```

and the run the game again. It is still a bit fast, but we can easily go to the subroutine again (and again) by changing the line to

```
245 GOSUB 1020;GOSUB 1020;GOSUB 1020
```

This speed seems about right, but the display is pretty hard to read. Add

```
162 PRINT £12
212 PRINT £12
```

and run it, and see how much easier it is to read.

Now in a few of the games you've played with the computer so far, while developing this program, you may have noticed that if you stood on, say, 19, not being willing to risk going over 21 and automatically losing, and the computer got to 19, it stayed there, rather than try and beat you. Before you read how we've done it, work out a line to allow the Atom to make the right decision in this situation. You'll need to use the 'inverse slash' (which means OR as you'll see in your manual, on page 15 of the second edition). We will show the inverse slash as <u>/</u>. It is on the keyboard between DELETE and @. You'll also need to use the ampersand (&) which can be used in Atom BASIC for the logical operator AND. You'll find the ampersand on shift 6.

The Atom must look at its own total (C) and compare this with the player's total (P) and if C is less than P, <u>or</u> C is equal to P <u>and</u> P is less than 22, the Atom must roll the die again. Try and work out what the line could read before you read our version. The new line will replace the current line 250.

```
250   IF(C<P)/(C = P)&(P<22) GOTO 210
```

If you run this program as it is, you'll find the Acorn just keeps a-rollin' dem bones until it beats you, or busts (that is, goes over 21). We said at the beginning that five rounds make up a game, and the winner would be the player who won the most games out of five. So we need to make the whole thing run on for five games, and for this we need a FOR/NEXT loop. A loop of this type simply counts the number of times you execute the program between the opening of the loop (the FOR statement) and the closing of the loop (the NEXT) going back to the FOR line over and over again until it has been through it the specified number of times. Add

```
80    FOR E = 1 TO 5
980   NEXT E
```

If you ran this, you'd find the Atom would play five games, but would have no idea of what was happening from game to game within the round. We need to add a routine to count the result of each game. You may have wondered what the 500 in line 200 (IF P>21 GOTO 500) when you were entering it. 500 will be the start of the routine which (a) works out and prints who has won the game and (b) adds to the score of the winner. We need to have variables to assign the result of each game. We'll use A (for Atom) and H (for human) to store the games' results.

```
10    A=0;H=0
500   IF(C=P)&(C<22) P."ITS A DRAW";A=A+1;H=H+1
510   IF(P>21)/((C>P)&(C<22)) P."I WIN";A=A+1
520   IF(P>C)& (P<22)/(C>21) &(P<22) P."YOU
      WIN";H=H+1
```

LINE 500 READS: If C equals P and C is less than 22 then it's a draw

LINE 510 READS: If P is greater than 21 or C is greater than P and C is less than 22 then the Atom wins

LINE 520 READS: If P is greater than C and P is less than 22 or C is greater than 21 and P less than 22 then player wins

If you run this, you'll find you don't even get to read the "I WIN" or "YOU WIN" before it vanishes, so you need to add a loop to slow it down for long enough to read. This gives us a chance to examine a new Atom function - WAIT. Instead of just GOSUB 1020 a number of times, we can add

```
970     F.Z=1TO10*60;WAIT;N.
```

This gives a delay of 10 seconds. If you wanted a delay of, say, 50 seconds, then the first part of the line should read

```
FOR Z = 1 TO 50*60
```

As you can see, WAIT is a very useful Atom function. The only thing we need to add to the program is the tally after each round. Work out your own display to do this, then compare your version with ours. This is the way we did it.

```
600     PRINT "AT THE END OF GAME "E'
610     P.'   '   '
620     P."MY SCORE     YOUR SCORE"'
630     P."  "A"          "H'
```

This game is quite satisfactory as it stands but you might enjoy 'dressing it up' a bit. Here is a complete listing of the program, which you can compare with your final version of the game.

```
10      A=0;H=0
80      FOR E=1TO5
90      @=0
100     PRINT £12
```

```
110     P=0;C=0

120     PRINT "ENTER 1 TO ROLL THE DIE,"'
130     PRINT "OR 2 TO STAND"'
135     PRINT "THIS IS GAME "E'
140     INPUT B
150     IF B = 2 GOTO 200
160     GOSUB 1000
162     PRINT &12
165     PRINT "YOU ROLLED A "D'
170     P=P+D
180     PRINT "YOUR TOTAL IS "P'
190     GOTO 115
200     IF P > 21 GOTO 500
210     GOSUB 1000
212     PRINT &12
213     PRINT "THIS IS GAME "E'
215     PRINT "YOU STAND ON "P'

220     PRINT "I ROLLED A "D'
230     C=C+D
240     PRINT "MY TOTAL IS "C'
245     GOSUB 1020;GOSUB1020;GOSUB 1020
250     IF (C<P)/(C=P)&(P<22) GOTO 210

500     IF (C=P)&(C<22)PRINT "IT'S A DRAW "';A=A+1;
        H=H+1
510     IF (P>21)/(C> P) &(C<22) PRINT "I WIN"';
        A=A+1
520     IF (P>C)&(P<22)/(C>21)&(P<22) PRINT
        "YOU WIN"';H=H+1

600     PRINT "AT THE END OF GAME "E'
610     PRINT';PRINT';PRINT'
620     PRINT "MY SCORE  YOUR SCORE"'
630     PRINT "   "A"       "H'
970     FOR Z=1 TO 10*60;WAIT;NEXT Z
980     NEXT E
981     PRINT &12
982     PRINT "I WON "A" GAMES""AND YOU WON "H'
985     IF A = H PRINT "SO IT'S A DRAW"'
```

```
986     IF A > H PRINT "SO I WIN"'
987     IF H < A PRINT "SO YOU WIN"'
990     END
1000    D=ABSRND%6 + 1
1020    FOR Z=1 TO 1000;NEXT
1030    RETURN
```

There are a number of ways you can shorten this program using the abbreviations given in the appendix, and by joining some lines together into multi-statements. Here's one shorter version.

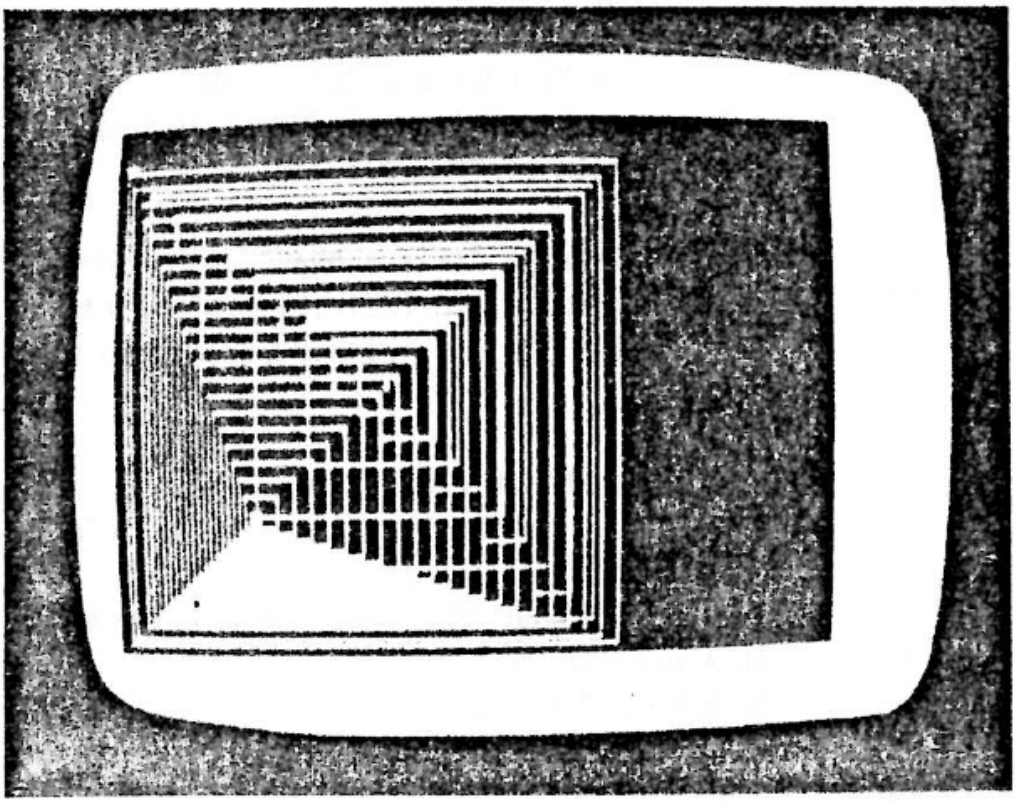
```
10      A=0;H=0
80      F.E=1TO5;@=0;P. ß12;P=0;C=0
120     P."ENTER 1 TO ROLL THE DIE,"'"OR 2 TO STAND"'
135     P."THIS IS GAME "E';IN.B;IFB=2 G.200
160     GOS.1000;P. ß12;P."YOU ROLLED A "D'
170     P=P+D;P."YOUR TOTAL IS "P';G.115
200     IF P> 21G.500
210     GOS.1000;P. ß12;P."THIS IS GAME "E'
        "YOU STAND ON "P'
217             P."I ROLLED A "D';C=C+D
220     P."MY TOTAL IS "C';GOS.1020;GOS.1020;GOS.1020
250     IF (C< P)/(C=P)&(P< 22)G.210

500     IF(C=P)&(C<22)P."IT'S A DRAW"';A=A+1;H=H+1
510     IF(P> 21)/(C> P)&(C <22)P."I WIN"';A=A+1
520     IF(P> C)&(P <22)/(C >21)&(P <22)P."YOU WIN"';
        H=H+1

600     P."AT THE END OF GAME "E'   '   '   '
610     P."MY SCORE  YOUR SCORE"'"   A"       "H'
970     F.Z=1TO10*60;WAIT;N.;N.
981     P. ß12;P."I WON "A" GAMES"'"AND YOU WON "H'
985     IF A=H P."SO IT'S A DRAW"'
986     IF A>H P."SO I WIN"'
987     IF H<A P."SO YOU WIN"'
990     END
1000    D=A.R.%6+1
1020    F.Z=1TO1000;N.;R.
```

BUILDING A LIBRARY

Resist the temptation to put all your programs - one after the other - on a long cassette. The time you'll waste while waiting for the Atom to find a particular program is not worth the money you'll save on cassettes.

Get a set of C-12 cassettes. Put just one program on each cassette, with two copies of the program on each side. This may sound a waste of time, but you are then covered if the start of your tape gets mangled, one of the copies will not load, or you accidentally wipe one side. Put the load name on each side of the cassette, and on the cardboard outer.

You'll find this makes it very easy to find the program you want, and as your library builds, it will give you quite a feeling of accomplishment to see all those programs ranked side by side.



WRITING PROGRAMS

Most books on programming suggest you start by drawing up a 'flowchart' -- a pretty combination of circles, diamonds and slanting rectangles -- which sets out the path and operations the computer will follow to execute the program. In theory this is fine, but in practice, especially when using a computer like the Atom which can quickly pinpoint errors, and will not 'self-erase' if you've made a programming mistake, the time and trouble involved is probably not worth it.

It is, however, essential to know exactly what you want the computer to do before you start creating a new program, even if you're not quite sure how you are going to get the Atom to carry out the task.

Sometimes a rough sort of flowchart -- just the main steps the Atom will take, linked by lines and loops -- will help to clarify your thinking. This is also a good way of spotting potential problems (such as an infinite loop) or of not specifying the computer's task exactly.

We generally work out the 'core' of a program on paper, then enter it into the Atom, starting at about line 100 to leave room before this line to assign variables, set up arrays and the like. If you're working on a fairly simple game, or are adapting one from a magazine or book, it is just as well to work directly on the Atom, but you could keep a notebook handy to record things like the fact that, for example, string $A is for the player's name.

If you've got added memory, don't let its presence lead you into sloppy programming habits. It is very easy to set up a long and clumsy set of IF statements which really should be replaced by an IF then a GOSUB. When you have memory to spare, it often seems too much trouble to bother cleaning up your programs. GOTO statements can

cover a multitude of situations which arose because you did not give sufficient thought to the maximum line number you would need. When you examine a listing, be particularly suspicious of any *non-conditional* GOTOs.

The best thing you can add to a longer program is the element of surprise. If you can include situations which do no occur every time a game is played, you'll ensure the game will remain interesting for a much ionger time than would be the case if every situation was triggered every time a game was run.

You can speed up slow-running games by using labels instead of line numbers for GOTOs and GOSUBs, and by placing often-used subroutines at the start of the listing. The program will run fractionally faster because the computer must search through the whole listing when it comes across a GOTO or GOSUB command. If the subroutine is near the start of the program, the Atom will not have to go through, say, the instructions every time it searches for a subroutine.

You can make the first line of the program GOTO some line number to jump over the subroutines. Variables which must be assigned at the start of a game can actually be listed right at the end of a program, ending with a GOTO leading back to the start of the game part of the program.

Another way of improving programs, and of making them more interesting to players for a longer time, is to add a "degree of difficulty", such as in the GRAND PRIX SPECIAL and the MULTIPLICATION QUIZ programs. Make sure that this feature really does increase the difficulty of the game, and ensure that -- even at the highest level of play -- the final score is reasonably attainable.

You can add further interest to games by awarding points, or scores, ratings or whatever that are genuinely related to the speed and skill the player demonstrated. A further twist is to award a 'rank' (like 'star fleet captain'

or 'incompetent fool') to the player, depending on how well he or she did. Points and ranks ensure a player remains interested in a game for a longer time, as the player will try to beat his or her best score or ranking.

Keep these points in mind when improving -- as you no doubt will -- the programs in this book, or when writing your own programs.

Before we move on, let's play a few more games.

NICOMACHUS

Essentially only the algorithm of the next program is supplied, so you can elaborate it to your heart's content.

Nicomachus, so we are told, was a mathematician and authro who wrote a spell-binding book with the gripping title of *Arithmetica* around the year 100. In this book is the mathematical puzzle used in this program. You think of a number between one and 100, and divide it successively by 3, 5 and 7, entering the *remainder* after each division. Line 30 then reconstructs your number from this information.

```
5       P.$12
10      P."THINK OF A NUMBER (1 TO 100)"'
20      IN."/3"Z;IN."/5"P;IN."/7"Q
30      M=70*Z + 21*P + 15*Q
40      F.K=1 TO 3
50      IF M >105  M = M - 105
60      N.
70      P."YOUR NUMBER WAS "M'
80      END
```

LABYRINTH

This is a very silly program which is included to give you the raw outline within which you can create an ADVENTURE program of your own. Enter this, play it a few times for a giggle, and then work within the framework to create your own masterpiece. If this was a computer games book that took itself too seriously, you'd get an introduction to the following program something like this:

> *Descending deeper into the dank, deathly darkness went a solitary figure. Suddenly, in front of him, there appeared a bulbous priggle waving a flaming brand over its mis-shapen head. He hardly had time to blink before the priggle lunged at him, and whisked his wallet from his pocket. Chuckling in a mad, insensitive warble, the priggle scampered off down the tunnel and out of sight. With his supply of silver gradually dwindling the man wondered whether he would have enough to pay the Minotaur...if he ever got to the end.*

```
10    P."LABYRINTH"'
20    F.M=1TO5;P.';N.
30    DIM    F25;X=0;S=30;W=1;@=0
40    P."YOU ARE AT THE START OF A""LABYRINTH OF
      MANY TWISTING,"'
50    P."TURNING TUNNELS, YOU  HAVE""A SACK
      HOLDING 30 PIECES OF"'
60    P."SILVER, YOU MUST GET TO""THE END OF
      THE LABYRINTH WITH"'
70    P."AT LEAST 20 TO PAY THE MINOTAUR"'   '
80    P."PRESS RETURN";LINK#FFE3;P.&12
90    F.M=1TO60;WAIT;N.
100   F.M=1TO7;P.&46;N.;P."THIS IS MAZE/TUNNEL "   '
110   IF W=10G.1160
120   P."NUMBER "W" OF THE LABYRINTH"';P.'
130   P."(10 IS THE END)"'; P.';X=X+1
140   P."THIS IS CHALLENGE NUMBER "X'
150   IF S < 1  S=5
160   P.';P."YOU HAVE "S" SILVER PIECES";R=ABSRND%5+1;P.'
```

The next few lines determine the number and type of exits that face you. If you increase the range of the random number generated un line 180, you can add a few more 'imaginative' exits.

```
170   P."FACING YOU NOW ARE "R"   "
180   L=ABSRND%6+1
190   IF L = 1 P."DOORS"'
200   IF L = 2 P."PORTALS"'
210   IF L = 3 P."GATEWAYS"'
220   IF L = 4 P. "TUNNELS"'
230   IF L = 5 P. "DOORWAYS"'
240   IF L = 6 P. "EXITS"'
250   P.';IN."WHICH ONE WILL YOU TRY"A
260   F.M=1TO60;WAIT;N.;P.&12
```

After the Atom has decided if you've fallen down any holes, or whatever -- which it does with the GOSUBs in lines 270 and 290 -- it chooses one of a selection of insults.

```
270   IFABSRND%10=5 GOS.850
290   IFA=ABSRND%6+1 GOS.850
300   T= ABSRND%6+1
310   IF T = 1 P. "FOOL"
320   IF T = 2 P. "HELP"
330   IF T = 3 P. "IDIOT"
340   IF T = 4 P. "DUMBO"
350   IF T = 5 P. "SHAME"
360   IF T = 6 P. "GUESS WHAT"
```

And after the insults, a selection of horrors. Your imagination can probably add many more to await you as you make your way through the labyrinth.

```
390   P."YOU'VE WALKED IN ON A"';K=ABSRND%14+1
400   IF K = 1 P. "RODENTING RAT"'
```

```
410   IF K = 2 P. "WART-FACED WOGGLE"'
420   IF K = 3 P. "LIPSOID OCTOPUS"'
430   IF K = 4 P. "WACKED-OUT WIZARD"'
440   IF K = 5 P. "HAIRY FAIRY"'
450   IF K = 6 P. "SNOGGLE-GLOP"'
460   IF K = 7 P. "ZAPHROID NURDLE"'
470   IF K = 8 P. "HUNGRY DRAGON"'
480   IF K = 9 P. " PUSILANIMOUS SIDRANT"'
490   IF K = 10 P."WUMPUSHURKLE"'
500   IF K = 11 P."GRUNGLED DINCKER"'
510   IF K = 12 P."PRATLING GNAT"'
520   IF K = 13 P."BULBOUS PRIGGLE"'
530   IF K = 14 P."SMELLY SMURF"'
```

...and each of them is armed with a deadly variety of weapons. After the Atom has chosen a weapon for your adversary, it asks you which of your three weapons you will use.

```
550   E=ABSRND%10 + 1
555   P."ARMED WITH A  "
560   IF E = 1 P. "FLAMING BRAND"'
570   IF E = 2 P. "SHINING SWORD"'
580   IF E = 3 P. "POISONED NEEDLE"'
590   IF E = 4 P. "GOSUB-MACHINEGUN"'
600   IF E = 5 P. "BLUNT INSTRUMENT"'
610   IF E = 6 P. "LASER PISTOL"'
620   IF E = 7 P. "RED-HOT POKER"'
630   IF E = 8 P. "RAZOR-SHARP SCYTHE"'
640   IF E = 9 P. "MAGIC WAND"'
650   P.'  '  '
670   P."WHICH WEAPON DO YOU CHOOSE "'  '
680   P."A FLOATING POINT ROM (1)?"  '
690   P."A FOR/NEXT LOOP (2)?"'  '
700   P."OR A POKED ADDRESS (3)?"'
705   IN.B
710   C=ABSRND3+1;F.M=1TO60;WAIT;N.;P.&12
```

This next section determines the outcome of the *Big Confrontation*. Whether you win or lose, the generous Atom tells you your present whereabouts and your financial situation.

```
720   IFB=C P."YOU BEAT IT"';S=S+ABSRND2+1;P."AND
      HAVE "S'
730   IFB=C P."SILVER PIECES"';W=W+ABSRND3+1
740   IF W > 10  W=10
750   P.';IFB=C G.80
790   P."YOU WERE DEFEATED"';S=S-ABSRND3+1
800   IF S < 1   S = 1
810   P."AND HAVE BEEN LEFT WITH "S" SILVER"'
820   W = W - 1 - ABSRND%2;IF W < 1  W = 1
830   G.80
850   K=ABSRND%4+1
860   IF K = 1 GOS. 920
870   IF K = 2 GOS. 980
880   IF K = 3 GOS. 1000
890   IF K = 4 GOS. 1110
900   G.80
920   P.';P."YOU'VE FALLEN DOWN"';P."A  "';J=ABSRND
            %4+1
930   IF J = 1 P."TRAPDOOR..."'
940   IF J = 2 P."HOLE..."'
950   IF J = 3 P."CHUTE..."'
960   IF J = 4 P."PIT..."'
970   W=W-1;S=S-ABSRND%2-1;R.
980   P."A WALL OF FLAME ENGULFS YOU"'
990   W=W-1;S=S-ABSRND%2-1;R.
```

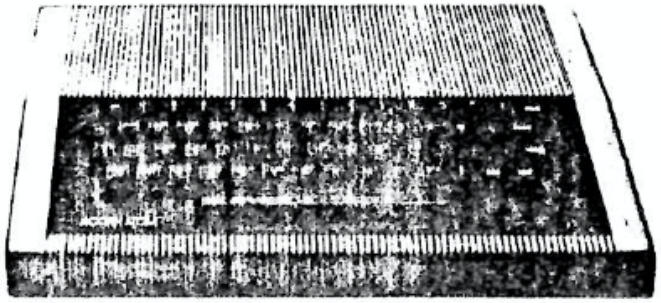
Of course, it is possible for nice things to happen. The Atom can choose from a well-stocked harem of princesses, one suitable for stroking your fevered brow (or you can make the whole harem full of hunky princes). Sometimes, instead of princes or princesses, you'll be offered cash.

```
1000  P."THE LOVELY PRINCESS  ";P=ABSRND%6+1
1010 IF P = 1 P."SEMOLINA"'
```

```
1020  IF P = 2 P. "AMBROSIA"'
1030  IF P = 3 P. "TAPIOCA"'
1040  IF P = 4 P. "FORMICA"'
1050  IF P = 5 P. "MACARONI"'
1060  IF P = 6 P. "BALALIKA"'
1070  P."SOOTHES YOUR FEVERED BROW"'
1080  S=S+ABSRND%4+1;W+W+ABSRND%2+1
1090 IF W>10  W = 10
1100  R.
1110  P."JOY OH JOY. A HOARD OF"'"SILVER. CHOOSE UP
      TO 5 PIECES"'
1120  P."BUT BE WARNED. THE MORE"'"YOU TAKE, THE
      MORE IT WILL"'
1130  P."COST YOU.  HOW MANY?"'
1140  IN.D;IFD > 5 G.1140
1145  S=S+D;W=W-D/2;R.
```

The final section determines what happens when you reach the end of The Labyrinth. Will you have enough money to pay the Minotaur and escape, or will you become Kentucky Fried Loser?

```
1160  P."YOU ARE AT THE END"'"DO YOU HAVE ENOUGH
      SILVER?""
1170  P."PRESS RETURN TO FIND OUT";LINK#FFE3;P.&12
1180  IF S<20 P."THE MINOTAUR EATS YOU"';P.&7;G.1180
1190  P."YES, YOU HAVE "S" SILVER"'"PIECES"'
1200  P."YOU HAVE WON"';F.J=1TO60;WAIT;N.;G.1200
```



TURNING THE TABLES

There are many, many programs in which the computer thinks of a number, and the human player has to guess it. This program turns the tables.

```
5     DIMB4
10    P.$12;P."MYSTICO"'   '
20    P."THINK OF A NUMBER BETWEEN"'
30    P."ONE AND 100 AND I WILL GUESS IT"'   '
40    P."PRESS RETURN TO START";LINK#FFE3
45    F.F=1TO60;WAIT;N.;P.$12
50    T=0;Z=100;Y=1;X=ABSRND%99+1;@=0
60    P."I GUESS "X'   '   '
70    P."RIGHT (R) OR WRONG (W)"
80    IN.$B;IF$B="R"G.180
90    F.F=1TO60;WAIT;N.;P.$12
100   P."MY GUESS WAS "X'   '
110   P."HIGHER (H) OR LOWER (L)"
120   IN.$B;F.F=1TO60;WAIT;N.;P.$12
130   IF$B="L"G.170
135   Y=X
140   S=Z-Y
145   X=Y+ABSRND%S
150   IFX=S+Y G.145
160   G.60
170   Z=X;G.140
180   F.F=1TO60;WAIT;N.;P.$12
190   P."AIN'T I CLEVER?"'
200   P."I GOT IT IN JUST "T" TRIES"'   '
210   P."COULD YOU FACE ANOTHER GAME";IN.$B
220   F.F=1TO60;WAIT;N.;P.$12
230   IF$B="YES"G.20
240   P."BYE BYE THEN WET BLANKET"';G.240
```

MASTERING THE GRAPHICS

The Acorn Atom has a very flexible graphics capability, as you can tell from the sample programs in your manual. But that very flexibility can make it a difficult area of the Atom to master. This section will take you through the graphics, step by step, and will hopefully demystify it to the extent where you'll be able to use it as effectively as the computer allows.

You'll have read in your manual that there are three basic plotting commands - PLOT, DRAW and MOVE. MOVE and DRAW are special cases of PLOT, so to avoid confusion we will assume, for the time being, that they don't exist.

Enter the following:

```
10      CLEAR 0
30      PLOT 13,32,24
140     END
```

A white square should appear about the middle of the screen. What this little routine has done is plotted a point in white at the position 32 across and 24 up, taking the bottom left hand corner of the screen as 0,0. Add the following, and run it:

```
40      PLOT 1,64,0
```

A row of points will appear, forming a line going to location 64. Change line 40 to PLOT 1,64,24 and run the program again. You should get a line, running up to the left hand corner.

List your program again. Line 10 CLEAR 0 has set the Atom in the lowest graphics mode. In this mode, there are 64 points across the bottom, counting from the left hand edge, and 48 moving up, counting from the bottom left hand corner.

When you use PLOT in a program, the first number after PLOT in the listing (that is the number 13 in line 30) tells the Acorn what you want to do, at the X,Y locations (the next two numbers in the line are X and Y). The 13 means 'plot a point in white at the position designated' while the 1 we've used in line 40 means 'draw a line in white relative to the last position'.

Add the following lines, which will automatically delete the 30 and 40 you now have, and try and predict what will happen when you press RUN.

```
30      X = ABSRND%64; Y = ABSRND%48
40      PLOT 13, X, Y
50      FOR A = 1 TO 300;N.
60      G. 30
```

A set of little squares should appear, being plotted at random positions. If you leave it running long enough, the whole screen will fill up. You can make quite an effective pattern-making program around this. Use ESC to stop the program, and enter the following (do <u>not</u> NEW it).

```
30      X = ABSRND%32; Y = ABSRND%24
40      PLOT 13,X,Y
50      PLOT 13, (64 - X), Y
60      PLOT 13, X, (48 - Y)
70      PLOT 13, (64 - X), (48 - Y)
80      F. A = 1 TO 50; WAIT; N.
90      G. 30
```

Run this for a while, and then add:

```
45      GOS. 80
55      GOS. 80
65      GOS. 80
75      GOS. 80
77      G. 30
90      R.
```

To make it run more quickly, now change line 80 to:

```
80      FOR A = 1TO 70;N.
```

After you've run this a few times, add the following, which creates quite an interesting effect:

```
35      IF RND > 0 G. 110
110     CLEAR 0
120     G. 30
```

Run this, and you'll get a somewhat startling effect which we won't describe. We'll leave it up to you to discover. When you're tired of this, change line 35 as follows:

```
35      IF RND%3 > 1 G. 110
```

Now, we'd like you to work out a way of getting particular points (rather than the whole screen) to vanish from time to time, so the program can be left running, and the screen will not end up eventually as a solid blob. After you've managed to do this (and - as a hint - PLOT 15,X,Y 'unplots' a blob) come back to the book and see how we did it.

```
110     PLOT 15, X, Y
120     PLOT 15, (64 - X), Y
130     PLOT 15, X, (48 - Y)
140     PLOT 15, (64 - X), (48 - Y)
150     G. 30
```

As 15 is the unplot command, from time to time the random X and Y generated in line 30 will coincide with a previously plotted blob, and line 35 will send the action to the unplot routine which will clear out a few white spots.

Try adding:

```
35      IF RND > 0 G. 105
105     F.A=1TO2
145     X = X + 1; N.A
```

You can play with this program to your heart's content. When you get sick of it come back to the book so we can move a little further in our exploration of the graphics. NEW the program, and enter the following:

```
10      CLEAR 0
20      X = 10; Y = 10
30      A = 1;  B = 1
40      PLOT 13, X, Y
50      C = X; D = Y
60      X = X + A; Y = Y + B
70      IF (X > 61)  / (X < 2) A = -A
80      IF (Y > 45)  / (Y < 2) B = -B
90      PLOT 15, C, D
100     G. 40
```

Run this, and you'll see a ball bouncing madly all over your screen, recoiling when it hits the edges. The program works as follows: Line 10 selects the graphics mode, line 20 positions the ball at the start (10,10). Line 40 plots the ball in position, and line 50 sets the other two variables (C and D) equal to this position. Line 60 works out the new ball position (either adding one to it, or taking it away, depending on which direction it is going).

Lines 70 and 80 change the values of A and B (which determine the direction of travel) if either X or Y indicates the ball has come to the edge of the screen. Line 90 makes the ball vanish and line 100 sends action back to line 40 to plot the

ball at its new position. As you'll see when you produce more complicated moving graphics programs, it is useful to have the moving thing unplotted for as short a time as possible, so it is advisable to have some way of holding the old position (in this case the variables C and D do this) while working out the new position.

You probably agree that the ball would look better if it was actually bouncing against something, so we'll add some 'walls' and introduce a new plotting number. Remember, we've used 15 which plots a white point at co-ordinates X, Y which are counted from the bottom left hand corner (X along the bottom, Y along the left hand edge); and 13 which unplots (or, if you prefer, plots a black blob) into the co-ordinates X, Y.

If the first number after the word PLOT is a 1 (as we said briefly at the start of this section) the Acorn will draw a line relative to the last point plotted.

Add the following:

```
11      PLOT 12, 0, 0      (moves the plotting position to 0,0)
12      PLOT 1, 0, 47      (draws a line up the left hand edge)
13      PLOT 1, 63, 0      (and across the top)
14      PLOT 1, 0, - 47    (down the right hand side)
15      PLOT 1, -63, 0     (and, finally, across the bottom)
```

You can optimise speed of operation by GOTOing or GOSUBing a *label* rather than a line number (and it saves problems if you change line numbers later). As maximum speed will become important as we add more to the program, change line 40 as follows (and remember we are using a lower case letter to equal the inverse Atom letter).

```
40z     PLOT 13,X, Y       and change 100:
100     G.z
```

To use this bouncing ball in a game, we need to be able to interact with it. Add the following:

```
17    E = 2; F = 2
41    PLOT 15, G, F
43    G = E
45    PLOT 13, E, F
46    IF ?#BU01 <> #FF; E = E - 1
47    IF ?#BU02 = 151 OR ?#BU02 = 135; E = E + 1
```

Line 45 plots a point just above the base line, and lines 46 and 47 check the keyboard to see if either the REPEAT key or the SHIFT key is being pressed. If REPEAT is being pressed, the point near to the base line moves to the right, and it moves to the left if SHIFT is being held down.

Run it now and make the blob move back and forth. Now, add line 65:

```
65    IF ABS(X - E) < 3 AND Y < 4   END
```

Now, to produce a kind of BREAKOUT, add the following:

```
16    FOR A = 5 TO 55 STEP 2;PLOT 13,A,45; PLOT
            13,A,43;N.A
18    FOR A = 5 TO 55 STEP 2; PLOT 13,A,41; PLOT
            13,A,39;N.A
```

Run this to see the bricks wiped out. Now we need the little blob at the bottom (the one controlled by SHIFT and REPEAT) to do something better than just stop the action. Make the following changes to make the program more like BREAKOUT.

```
65    IF ABS(X - 3) > 5 AND Y< 4 END
```

Run this, and you'll see you have to get your blob near the bouncing blob to keep the game going. We'll now change 65, and add a few more lines.

```
65    IF ABS(X - 3)>5 AND Y<4 GOSUBH
```

```
19    L = 20
110H  L = L - 1
120   IF L = 0 END
130   R.
```

This will allow you to 'miss' the ball four times. The game stops at the fifth miss. You've probably been thinking that the paddle should be wider. We'll add a bit of width to it now, but notice how this slows the whole game down tremendously.

```
41    PLOT 15,G,F;PLOT15,(G+1),F;PLOT15,(G+2),F;
            PLOT 15,(G+3),F
44    PLOT 13,E,F;PLOT13,(E+1),F;PLOT13,(E+2),F;
            PLOT 13,(E+3),F
```

For a bit of relief from all this learning, add the following for a moment: `75 IF RND > 0 A=A + 1` *Once you've seen the evil, drunken effect it has, delete 75 -- but keep the effect in mind, as you may be able to use it in a program sometime.*

To make the game slightly less predictable, add:

```
125   IF X<60 X = X + 2
```

Finally, now that you've seen how much extra statements can slow the program down, make the final changes to end up with a game worth playing:

```
41    PLOT 15,G,F
44    PLOT 13,E,F
```

Save this on cassette before NEWing your Acorn to continue exploring the graphics.

OLYMPIC SWIM

This next program uses plot and 'unplot' to create a very bad facsimile of a swimming pool, with four little

swimmers racing up and down it for their lives. Note that this program stores the position of the swimmers in an array AA and unplots them with the array GG.

```
10    CLEAR 0; @ = 0
20    DIM  AA(4)
30    DIM GG(4)
40    F. A = 1 TO 4; AA(A) = 0 ; N.
50    F. X = 0 TO 64; F. Y = 18 TO 31; PLOT 13,X,Y;
            N.Y;N.X
60    A = 20; B = 23; C = 26; D = 29
70    G. 120
80    PLOT 13, (ABS(64 - GG(1))),A
90    PLOT 13, (ABS(64 - GG(2))),B
100   PLOT 13, (ABS(64 - GG(3))),C
110   PLOT 13, (ABS(64 - GG(4))),D
120   PLOT 15, (ABS(64 - AA(1))),A
130   PLOT 15, (ABS(64 - AA(2))),B
140   PLOT 15, (ABS(64 - AA(3))),C
150   PLOT 15, (ABS(64 - AA(4))),D
160   F. H = 1 TO 4; GG(H) = AA(H);N.H
170   IF (AA(1) > 127) & (AA(2) > 127) & (AA(3) > 127)
            & (AA(4) > 127) G.200
180   F. E= 1 TO 4; AA(E) = 1 + AA(E) + ABSRND%2;N.E
190   G.80
200   P. "A = "AA(4)' "B =  "AA(3)'
210   P. "C = "AA(2)' "D =  "AA(1)
220   END
```

In this program, line 50 plots the 'swimming pool' across the full width of the screen. Lines 80 to 110 unplot the swimmers, and lines 120 to 150 plot them in their new positions. Line 160 sets the unplot positions, line 170 looks to see if any of the swimmers have reached the finish line, and 180 increments each swimmers' position (by adding one, plus a random figure -- either zero or one -- to the array which holds that swimmer). Once you understand how the program works, try to rewrite the plot (120 - 150) and unplot (80 to 110) routines in a much shorter way. They have been listed in this long way in the book to make it clear what was happening.

A-PEEKing and a-POKEing

If you wish to do more than just PLOT or UNPLOT points, you can POKE directly into the screen. PEEK and POKE are discussed in more detail elsewhere in this book (see the section starting on page 79). This part of the book will just look at a simple form of POKEing. Enter the following:

```
10    CLEAR 0
20    ?#8000 = ABSRND%255
30    F.A= 1 TO 20;WAIT;N.
40    G. 20
```

Run this, and you'll see an endless series of characters appearing in the top left hand corner of the screen. #8000 is the HEX address of the top left hand corner of the screen. Any other address on the screen is given by #8000 + 32*L + C, where L is a line number between 0 and 15, and C is a column from 0 to 31.

We'll now try and develop a program using POKE. To produce a similar effect to OLYMPIC SWIM, enter the following program:

```
10    CLEAR 0
20    Z = ABSRND%640
30   Z?#8000 = 127
40    G. 20
```

This will eventually fill the screen with randomly POKEd character 127's. NEW this, and enter:

```
10    CLEAR 0
20    A = 0
30    B = A
40    B?#8000 = #40
50    A = A + 1 + ABSRND%2
60    A?#8000 = #7F
70    F. J = 1 TO 10;WAIT;N.
80    G. 30
```

Run this, and you'll see the cursor gradually work its way down the screen (note that #40 is the unPOKE character). To make it a 'race', add the following:

```
25    C = 64
35    D = C
45    D?#8000 = #40
55    C = C + 1 + ABSRND%2
65    C?#8000 = #66
```

When you run this, you'll see two tiny things have a race (of sorts) across the screen. Change the #7F in line 60 to #EE and you'll have two tiny space ships racing across the screen. If you want to see how fast they can go, delete line 70, or replace it with just WAIT;WAIT; WAIT. We will not elaborate the program but leave it up to you to devise a space game of your own.

Having run this, you should now have a pretty good idea of how to create your own 'moving graphics' games. As you've seen, POKE/unPOKE graphics move much, much more quickly than the PLOTted ones we've been using. Perhaps we could write a kind of BREAKOUT using POKE which would be more challenging to play. Enter the next bit of code to get a ball on the screen.

```
10    CLEAR 0
20    A = 1;B = 1;L = 8;C = 8
30    Z = 32*L + C
40    Z?#8000 = 48
1000  END
```

Now, a mechanism to move the ball:

```
60    IF (L<2) / (L > 13)  A = - A
70    IF (C<2) / (C > 29)  B = - B
80    L = L + A;C = C + B
```

Run this, and you'll see the 'ball' gradually cover the whole screen. We need to unPOKE the 'old' ball, so we need to store its position and then unPOKE it after

the new position has been calculated. Add the following:

```
5     D = 8; E = 8          (initial position to unPOKE)
35    X = 32*D + E
55    X?#8000 = 32
50    D = L; E= C
```

Run this, and see just how fast the ball is moving. If you like, change line 30 to 30aZ=32*L+C and change line 90 to 90 G.a

We may as well put a frame around the screen, using the same routine as we did in the PLOT BREAKOUT.

```
11    PLOT 12,0,0-PLOT1,0,47;PLOT1,63,0;PLOT1,0,-47;
            PLOT 1,-63,0
```

You can run this, and leave the poor ball bouncing frantically in its box for ever. Alternatively, you could give it something to knock down.

```
12    F.A = -1 TO 24 STEP 2; A?#8064 = #FF;N.A
```

Run this and note (a) how quickly the screen is set up and (b) the effect of line 12. If you want the ball to make a noise when it hits the sides, add:

```
7 M = #B002
```

and add at the end of lines 60 and 70 ;?M=?M:4

If we're going to play BREAKOUT, we need a paddle:

```
13    P = 449
45    P?#8000 = #43
```

This will make the paddle appear near the bottom left hand corner of the frame, but we need a mechanism to make it move:

```
14    R = P
46    R = P  (to remember its position to unPOKE it)
```

The next line checks to see if the SHIFT key is being pressed and also to see if P is greater than 449. If both

these conditions are true, the paddle is moved one square to the left.

```
47    IF(?#B001&#80 = 0)&(P  > 449); P = P - 1
```

The next line does the same for the REPEAT key and a maximum allowable value of 477 for P.

```
48    IF(?#B002&#40 = 0)&( < 477); P = P + 1
```

Check that this is working before continuing. We must now work out a way of knowing whether the ball has hit the paddle or not, and we'll do this the way we did in PLOT BREAKOUT by checking the co-ordinates of the ball and the paddle. If the ball is on line 13 (that is, L equals 13) and its column (C) is within two either way of the column position of the paddle (P) we will allow the ball to bounce up again. If not, the ball will disappear. We need to change line 60.

```
60    IF((L> 13)&(ABS((P - 449) - C) < 5)) ∠ (L<2)
          A = -A;?M=?M:4
```

When you run this, you'll notice that, sometimes, if you don't get your paddle near the ball when it gets to the bottom of the screen, the ball will vanish, only to reappear after a random interval, shooting up violently from the bottom. However, this is rather uncontrolled, so we can add a line to bring it under control and allow for, say, five balls to 'vanish' before the game ends.

```
9     W = 5
65    IF (L> 15)&(ABS((P - 449) - C) > 4) GOS.Q
100Q  F. I = 1 TO 2000;N.I
110   C = 4 + 2*(ABSRND%10)
120   A = - A
130   W = W - 1
140   IF W >  0 R.
150   P. "GAME OVER"
```

And, just so you'll know how you're going, we'll POKE the score (that is, the value of W) into the screen.

```
32    ?#80A0=48 + W
```

You'll probably find this game too easy to play after you've had a few rounds with it, so increase its difficulty by changing line 60 tso that it reads less than 3, rather than less than 5; and line 65 so that it reads greater than 2 rather than greater than 4. You might like also to change the shape of the ball. Try putting a 42 in place of the 48 in line 40, or if you want a really wild effect, change line 40 to the following:

```
40    Z?#8000 = 33 + RND%10
```

You can make the paddle respond more quickly to your touch by changing the - 1 in line 47 to - 2 and the + 1 in line 48 to + 2. You can also add another row of 'bricks' by adding this line:

```
15    F. A = - 1 TO 24 SIEP 2;A?#80A4= #FF; N.A
```

Here's a final listing of the whole program.

```
5     D = 8;E = 8
7     M = #B002
9     W = 10
10    CLEAR 0
11    PLOT 12,0,0;PLOT1,0,47;PLOT1,63,0;PLOT1,0,-47;
      PLOT1,-63,0
12    F.A=1TO 24 STEP2;A?#80A4=#66;N.A
13    P = 460;R = P
15    F.A= -1TO24 STEP2;A?#8064=#59;N.A
20    A = -1;B=1;L=8;C=8
30A   Z=32*L+C
32    ?#80A1=48+W
33    X=32*D+E
35    X?#8000=32
40    Z?#8000=33+RND%10
41    R?#8000=32
45    P?#8000=#43
```

```
46    R=P
47    IF(?#B001&#80=0)&(P>449);P=P-2
48    IF(?#B002&#40=0)&(P<477);P=P+2
50    D=L;E=C
60    IF(L>13)&(ABS((P-449)-C)<3))/(L<2) A=-A;?M=?M:4
65    IF(L>15)&(ABS((P-449)-C)>2))GOS.Q
70    IF(C<2)/(C>29)B = -B;?M=?M:4
80    L = L+A;C=C+B
90    G.A
100Q  F.I=1TO 2000;N.I
110   C=4+2*(ABSRND%10)
115   A=-A
120   W=W-1;P.87
125   IF W>0 R.
140   P."GAME OVER"
150   END
```

Of course, you can compress this program considerably if you like by making the single lines into multi-statement lines.

BACK TO PLOTTING

Let's leave the POKEing of graphics for a moment and return to PLOTting. There are nine graphics modes, five of which are available on the unexpanded Atom (the other four are for colour which needs machine code software or the external ROM to give you the COLOUR statement).

Enter and run the following, which plots a graph in the CLEAR 1 mode.

```
10    CLEAR 1        (sets up a screen of 128 X 64 points)
20    F.X= 1 TO 128
30    Y = X/2
40    PLOT 13,X,Y
50    WAIT;WAIT;WAIT
1000  END
```

Now, without returning to CLEAR 0 so you can see the listing, type in 10 CLEAR 2 (which you'll have to do 'blind'), press

RETURN, then run the program. Note the effect of the finer graphics (128 X 96). Do this with CLEAR 3 then CLEAR 4.

Enter, directly, CLEAR 0, then LIST and change the program so it reads as follows:

```
10    CLEAR 4
20    FOR Y = 1  TO 192
30    X = Y/2
40    PLOT 13, X, Y
50    WAIT;WAIT;WAIT
60    N.
1000  END
```

And this, just to show how fine the resolution can be:

```
10    CLEAR 4
20    F. X= 1TO 200
30    Y = x*x/200
40    PLOT 13, X, Y
50    WAIT;WAIT;WAIT
60    N.
1000  END
```

Now this, to show the effect of different commands:

```
10    CLEAR 4
15    PLOT 4,0,0
20    F. X= 1 TO 22
30    Y = X*X/30
35    WAIT;WAIT;WAIT
40    PLOT 1, X, Y
50    N.
1000  END
```

Delete line 35 and run it again. Then, to see why PLOT 1 is so fast, add 35 F.J=1 TO 20;WAIT; N.

When you're tired of this, erase the contents of the Atom's memory, and enter and run the following:

```
5   CLEAR4
10  A=9;B=180
20  Z=13
30  F.C=A TO B;PLOTZ,C,A;PLOTZ,C,B ;PLOT Z,A,C;
    PLOTZ,B,C;N.
100 A=A+4;B=B-4
170 IFB -A>=1 G.30
180 IF Z = 13  Z = 15;A=9;B=180;G.30
190 G.10
```

After you've run this, change line 5 to CLEAR 3 and then, after running it, to CLEAR 2, and see the effect these changes have.

Now, change line 5 back to CLEAR 4, and change line 100 so it reads:

```
100 A=A+6;B=B - 1
```

Then, after you've run this, add:

```
100 A=A+4;B=B - 1
```

Now, to turn your Atom into a Cubist master artist, enter and run the following:

```
5   CLEAR4
10  A=A.R.%6+5;B=A.R.%30+151
12  D=10+A.R.%60;E=20+A.R.%90; Y=1+A.R.%9;
    X=1+A.R.%9;M=3+A.R.%Y
15  IF E<D  G.12
20  Z=13;IFR.>0 Z=15
30  F.C = A TO B; PLOTZ,C,A;PLOTZ,C,B;PLOTZ,A,C;
    PLOTZ,B,C;N.
110 A=A+Y;B=B-X
130 IF R.>0 Y = M;X=(M+X)/2
```

```
170  IF B - A>= 1 G.30
175  X=4+A.R.%4;Y+1+A.R.%12
180  IF Z = 13 Z= 15;A=A.R.%10;B=A.R.%60+122;G.30
190  G.10
200  END
```

ADDING COLOUR

To get a totally different effect, whether you have a colour card or not (but you will need the floating point ROM), add the following:

```
11   COLOUR RND
```

As you'll see this gets rid of the 'black background' although you may have to adjust your brightness and contrast controls to see the 'colours'.

There are four colour graphics modes (1a, 2a, 3a and -- not surprisingly -- 4a) which you get by entering CLEAR something (the number without the 'a') and then the COLOUR statement.

Value after the word COLOUR	*Colour television*	*Black and white television*
0	*green*	*grey*
1	*yellow*	*white*
2	*blue*	*black*
3	*red*	*black*

Once you've told the Atom which colour you want, it will DRAW in that colour, will PLOT lines and points in that colour and will 'unPLOT' to the background colour. Invert (PLOT commands 2, 6, 10 and 14) will produce a different colour to the specified colour.

This next little routine can produce rather startling results.

```
5     REM BROKEN GLASS
10    G =A.R.%4+1
20    CLEAR G
30    COLOUR R.
40    MOVE 40,(40+R.%4)
50    PLOT 1,(R.%100),(R.%100)
60    PLOT 2,(R.%20),(R.%20)
70    PLOT 5,(R.%10),(R.%10)
80    IF R.%30 < 28  G.30
90    F.J=1TO70;WAIT;N.;G.10
```

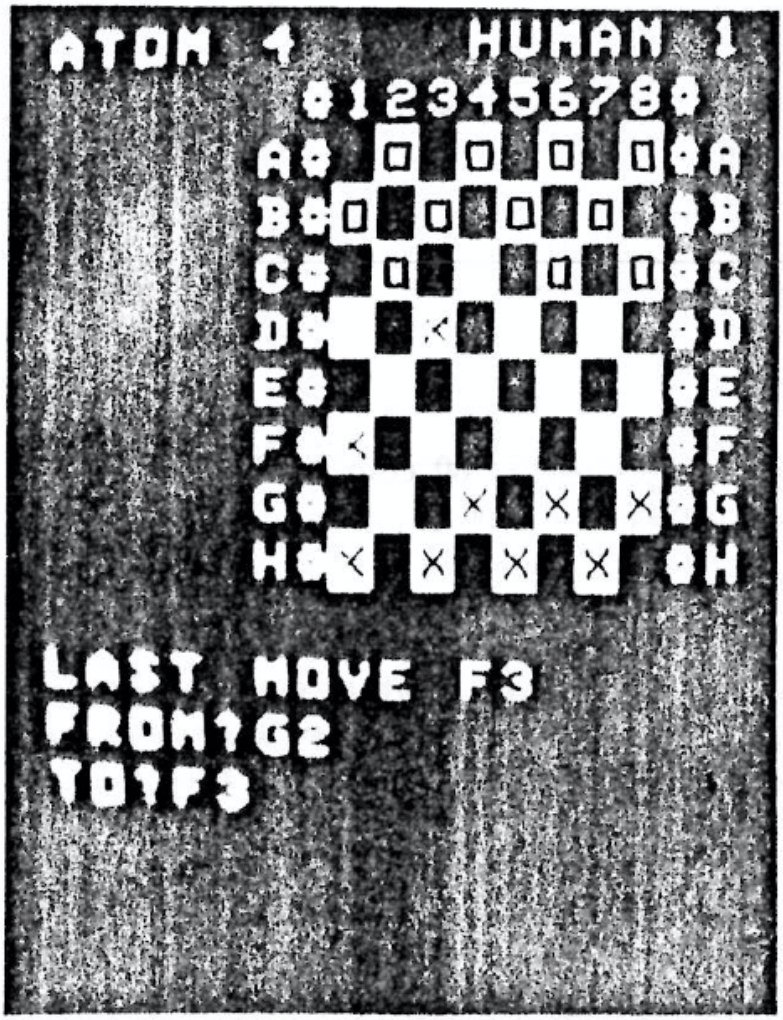


TABLE AND GRAPH

This next little routine creates a table of values in accord with the formula you place in line 60, then plots the values. You can easily dispense with the plotting part of it if you like by deleting lines 95 to 120. You will have to experiment with the *scale* used to get the most effective plot of your formula.

```
5     CLEAR 0
10    INPUT "LOWEST VALUE (X) TO PLOT" ' "AGAINST"A
20    INPUT "HIGHEST"B
40    DIM CC(B)
50    F. D = A TO B
60    CC(D) = ....your formula here...
70    P. D     CC(D)'
80    N. D
90    F. J= 1 TO 100; WAIT; N.
95    CLEAR 0
100   FOR D = A TO B
110   PLOT 13, D, CC(D)

120   N.
130   END
```

Try it with the following: 60 CC(D) = D*D/100 and A equal to one, B equal to 60.

LIFE

In this program, which uses PLOT in CLEAR 1 mode, we make use of a numbered grid, with the same numerical relationship between adjoining squares. This is explained in some detail in the section on draughts.

LIFE was invented by John Conway of Cambridge University in 1970. The game, which simulates the birth, growth and death of a cell colony, produces fascinating effects.

Each cell in the colony is born, survives or dies according to Conways rules:

- *Each cell on the grid (which is drawn on the outside of a cylinder in our case) has eight neighbours*

- *Every cell with two or three neighbours survives to the next generation*

- *If there are three, and only three neighbouring cells, a new cell is born*

- *Any cell with four or more neighbours dies from overpopulation*

Because of the way this program checks each cell (from the bottom of the area on which they are 'growing') the rules are not always followed exactly. Once you've seen this in action, you're invited to modify the rules slightly and see what changes this has on the way the colony evolves.

```
5     M = 0
10    CLEAR 1
20    DIM AA(255)
30    DIM EE(8)
40    FOR Z = 11 TO 255
50        AA(Z) = 0
60    NEXT Z
70    EE(1) = 15; EE(2) = -15; EE(3) = 1; EE(4) = -1
80    EE(5) = 16; EE(6) = -16; EE(7) = 14;EE(8) = -14
150   FOR Q = 1 TO 70;B=ABSRND%218+18;AA(B)=2;N.
200   K=1
210q  Y = 18; X = 46
230   FOR R = 11 TO 235
240   H = 0
250   IF (K=1)/(R<18) G.s
260   FOR G = 1 TO 8
270   IF(AA(R + EE(G))=2) H = H + 1
280   N. G
```

```
285   IF H<2  M = 0
290   IF (H> 1)&(H <4)  M=AA(R)
300   IF H = 3  M = 2
305   AA(R) = M
310   M = 0
320s  IF AA(R) = 0 PLOT 15,X,Y
330   IF AA(R) = 2 PLOT 13,X,Y
335   X = X + 2
340   IF X = 76 Y = Y + 2; X = 46
360   NEXT R
370   K = 0
380   GOTO9
```

As I said before the listing, this program makes use of a numbered grid. The grid, in this case, is 15 X 15, and the relationship between squares on the grid is:

-15 between squares on top of each other
-1 to the right
-14 above and to the right
-16 above and to the left

Once you've seen this program in action, you might like to write a version called MIRROR LIFE. In this, there is an 'inverse colony' to the right of the master colony. In can be done very simply by adding an extra plot statement to lines 320 and 330 (using the idea given for the pattern making program earlier in this section). If you feel really ambitious, you can make it a LIFE which is mirrored in three directions, so you have four colonies evolving at once. These can look very good indeed, especially if you select a higher graphic mode than the one given here.

PEEKING AT LIFE

Next we'll develop a version of LIFE using PEEK and POKE. You'll see this runs more quickly than the PLOT version and is also more flexible. You can choose any character you like for the cells. We'll use the asterisk (character #2A) but you can use anything you like. Like the preceding version, this LIFE starts off with a random (albeit biased) colony, but you can easily add a mechanism to enter your

own colony, if you prefer starting with a colony which is in a definite shape, or forms a pattern.

```
5     P.$12;M=0;G.1000
20e   F.J=1TO200;P=GG(J);P?#8000=#2A;N.
30q   J=20
40    DO;H=0
42    F.Z=1TO8;M=J+AA(Z);IF M?#8000=#2A;H=H+1
43    N.
45    IF(H=3)J?#8000=#2A
50    IF((H<2)/ (H > 3))J?#8000=32
60    M=0
80    J=J+1;UNTILJ=552
90    G.q
100   END
1000  DIMAA8,GG200
1010  AA1=32;AA2=31;AA3=33;AA4=1;AA5=- 1;AA6= - 31;
      AA7= - 32; AA8= - 33
1020  F.J=1TO25;L=A.R.%7+3;C=A.R.%10+5
1030  F.K=1TO2;GG(J*K)=32*L+G+K;N.
1032  F.K=3TO4;L=L-1;GG(J*K)=32*L+C+K/2;N.
1034  F.K=5TO6;GG(J*K)=32*L+C+K;N.
1036  F.K=7TO8;L=L-1;GG(J*K)=32*L+C+K/2;N.
1040  N.
1050  CLEAR0;G.e
```

You might like to try it with #2E as the 'cell' (this is the full stop). Once you've run this for a while, you can change to to sample the cells at random, rather than sytematically as above.

After a few minutes of frenzy, the colony will settle down quite satisfactorily. If you'd like your colony more spread out (which will lead to more cells dying in its early moments) change line 1020 to:

```
1020  F.J=1TO50;L=A.R.%9+4;C=A.R.%20+4
```

As you'll see from running the program, the longer you can leave it, the more satisfactory the colony becomes. If you

wish to see chaos grow from order (a most laudable aim) make the following changes:

```
20eF.J=20TO250;J?#8000=#2E      (or #2A)
Delete    1030, 1032, 1034, 1036, 1040
```

IMPLODERS

We'll continue our exploration of the graphics potential of the Atom by trying to write another game using, in the main, PEEK and POKE. So we can have sound with this, we'll use the routine to produce music given in the manual. Let's enter that first.

```
5     P.$12;P.$21
10    DIMVV(4),P(-1);L=#B002
30    ;:VV0 LDA L;LDY #81;:VV1 LDX #80;:
      VV2 DEX;BNE VV2
80    EOR (this is the letter O, not zero) @4;STA L;DEY;
      RTS; ]
85    P.$6
87    G.210
90    Z=A.R.%30+8
100?#81=255
110?#80=Z;LINK VV0
200   RETURN
```

Check that it is working by adding:

```
210   GOS.90
220   G.210
```

If it is working, you'll hear a munchkin chorus quiggling away in your sound box. Stop the program with ESC and delete lines 210 and 220. Now add:

```
82    Y=1;P=449
```

In CLEAR 0 there are 64 points across the bottom of the screen and 48 counting up from the bottom.

```
210  CLEAR0
220  F.A=1TO128;P=#EE+A.R.%22;N.
230  P?#8000=#C7
235  P=P+Y;T=P-Y
240  IF(P < 450)/(P > 478)Y=-Y
870  WAIT;WAIT;WAIT
880  T?#8000=52
900  GOS.90;G.230
```

Run this, and you should see a strange white object appear at the top of the screen, and a little thing should start beeping along the bottom. What we want to do is use the thing at the bottom -- which we'll call our base -- as a firing platform from which to shoot little missles up to the top. We need some way to fire. Delete line 870 and add:

```
250  IF?#B001&#80=0 GOS.q
260  G.880
500q END
```

Run this and once it is underway, press the SHIFT key and the base should stop. P - 448 is the approximate co-ordinate of the base across the bottom, We want to fire from the base to the top, so the laser bullet's position across the screen is 2*(P-448 - Y)+ 1. We'll call it X.

```
245  X=2*(P-448-Y)+1
```

The Y co-ordinate at the bottom of the screen is 0, so if we PLOTted at X,6 , we should see a point appear just above the base. Try it and see, by changing line 245 to:

```
245  X=2*(P-448-Y)+1;PLOT13,X,6
```

and to unplot it:

```
855  PLOT15,X,6
```

Run that, and make sure the little white square stays above the 'firing turret' of your base. When you're happy with that, read on. We now want a 'bullet' to fly across the screen from the gun.

```
274  W=6
500q PLOT13,X,W;PLOT15,X,W
520  IFW<48  W=W+1;G.q
540  R.
```

Run this, and from time to time, touch the SHIFT key. A tiny bullet should streak across the screen, wiping out a line of the odd shape at the top.

Now delete line 90, and change the Z in line 110 to a 1. Change line 900 to:

```
900  GOS.100;G.230
```

Run this, and you'll find the base moves much more quickly, but now only makes a clicking sound. Let us add a noise to the bullet:

```
510  ?#80=23;LINK VV0
```

Now although this makes a good sound, it slows things down a lot, so we can add:

```
505  IF W > 9 G.520
```

Try this. You still get a little burst of sound when you fire, but it doesn't stop the thing running. To actually see the bullet move, add:

```
500q PLOT13,X,W
502  WAIT
515  PLOT15,X,W
```

and change the 520 at the end of line 505 into 515.

Run this now, and fire the bullet a few times. When this game is complete, your whole aim will be to get the bullets right through the white space. It won't happen automatically like it does at this stage. The WAIT in

line 502 slows it down a bit too much, so delete line 502 and add:

```
507  WAIT
```

This ensures that the computer only WAITs if W is less than 10. Now to maximise the speed of the game so far, change 505 to:

```
505  IF W > 9 G,u
515u PLOT15,X,W          Make
260  G,e                 all
800e T?#8000=32          these
100j J?#81=255           changes
230k P?#8000+#c7         as
900  GOS,j ;G,k          well...
```

Delete line 895 for a new effect. Notice how much faster the program runs with these labels in the place of line numbers as destinations.

Now we're going to leave this program more or less as it is, so you can complete it as you choose. There are many, many games you can create from this point. We will give you the additional lines to create one game, but before you enter them, we suggest you save the program so far on cassette, then add our final lines to see what we've done with it. Then, come back to your saved version and write your own (doubtless better) game.

```
211  F=0;P.'
501  IF RND%9>7 G,z
505  IF((W>9)&(W<46)) G,u
512  IF W>45  G,z
540zR.
890  F=F+1
895  IF F=600 P.$7;END
```

When you run it, you'll find you have a limited time in which to try and get as many blocks through the obstacles to the other end. Many of them will fall short and stop

below the white thing at the top. You need to try and 'push' them up with additional bullets. When the game ends (which it does with an error code beep), add up the blobs and give yourself a score as follows: For every blob that got right through the white thing and is sitting on top, 50 points, minus 10 points for every blob left below the white thing.

From this scoring, you'll see it does not pay to just fire as many times as you can. It is better to try and push stray bullets up into the shape, but not to waste shots putting a bullet into a 'lane' where one is already sitting right on top. This will all be clear when you run the program.

Now there is an almost unlimited number of games you can create from the raw framework which you saved earlier, and from this game as it stands at the moment. For example, you could add a scoring mechanism which counted how many times you fired, and how many blobs got right through, and printed up this score at the end. One way of doing this is to add:

```
85   P.86;N=0;B=0
```

N will count the number of shots you fire (each costs you three points) and B will count the number of bullets which get through the shape (each worth 600 points).

```
500q PLOT13,X,W;N=N+3
506  IF W = 48  B = B + 600
895  IF F= 600 P.87;P."SCORE "B-N;END
```

WEAVING A TANGLED WEB

Let's examine plotting in a higher graphic mode than CLEAR 0. Enter and run the following program:

```
1    CLEAR0
5    GOS.a
```

```
20    PLOTK,X,Y
30    PLOTK,X,(64-Y)
40    PLOTK,(128-X),Y
50    PLOTK,(128-X),(64-Y)
60    G.5
1000  END
2000aX=A.R.%128+1;Y=A.R.%64+1
2100  K=13+A.R.%3
2200  IF(K<>15)&(A.R.%4 < 3)K=15
2500  R.
```

Run this, and you'll gradually see a pattern build up, then evolve, on a grid measuring 128 (along the bottom) by 64 (along the side). We'll move it into the top mode, CLEAR 4. Make the following changes:

```
1     CLEAR4
30    PLOTK,X,(192-Y)
40    PLOTK,(256-X),Y
50    PLOTK,(256-X),(192-Y)
2000aX=A.R.%256+1;Y=A.R.%192+1
```

You may have to adjust the brightness and contrast controls on your television to see the remarkably fine and detailed patterns which evolve.

We're going to work out our next program in graphic mode 1 and we suggest you then try and convert it (just as we've converted the preceding program into mode 2, 3 or 4. Doing this should increase your confidence with the graphics. Before you NEW the last program, add:

```
2300  IFA.R.%9=8 K=5
```

and run it for a while.

INTERGALACTIC GARBAGE COLLECTOR

This game uses CLEAR 1, and moves you around using SHIFT and BREAK like some of the previous games. you have a limited time to clean up as much space garbage (the

ubiquitous white blobs) as you can.

```
10   G.120
```

The next two lines read the keyboard.

```
20a  IF?#B001&#80=0 A=A-2*Y
30   IF?#B002&#40=0 B=B-2*X
```

These two lines put you in position.

```
40   A=A+Y;B=B+X
50   IF(A<1)/(A>127) Y= -Y
60   IF(B<1)/(B > 63) X= -X
```

Line 70 moves you from your previous position.

```
70   PLOT15,C,D
```

And line 80 gets your new position, ready for the next time it comes round to line 70.

```
80   C=A;D=B
90   PLOT13,A,B
```

The variable T counts the number of times you've been through the routine.

```
100  T=T+1;?#8000=T;IF T = 2000  END
110  G.a
120  A=64;B=32;Y=1;X=1
130  CLEAR 1
140  F.J=1TO50
```

This routine plots the 'garbage'.

```
150  PLOT13,(A.R.%128),(A.R.%64);N.
```

Line 160 sets the timer to zero.

```
160  T=0;G.a
```

When you run this, you'll find the SHIFT and REPEAT keys will alter your direction of travel. You simply have to try and clean up all the garbage, by running over it (and you must be spot on to clean it up) before you run out of time. Once you are confident with this game, add the following to the ends of lines 20 and 30:

```
;T=T+10
```

This will penalise you every time you press REPEAT or SHIFT.

PURSUIT

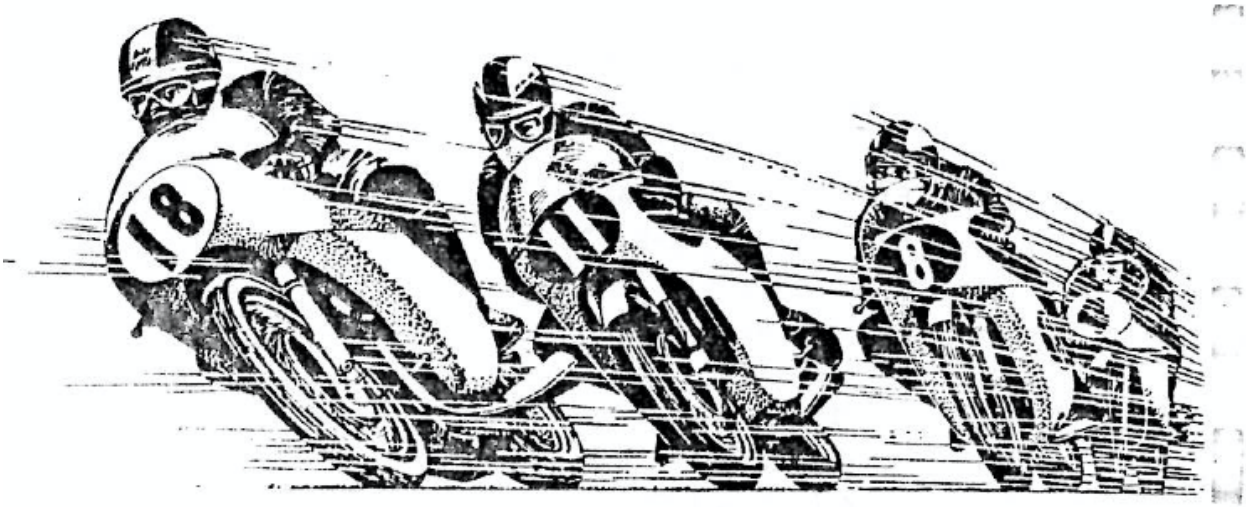
Before you read on, we'd like you to try and write a game, in graphic mode 3, which will have you moving about as in the preceding program, but this time you will be chasing another space ship. The game ends when you capture it. Try and write such a program, using the ideas we've discussed so far, before reading our listing.

```
10   P.&21
20   DIMVV(4),P(-1);L=#B002; [;:VV0 LDA L;LDY #81;
     :VV1 LDX #80
30   :VV2 DEX;BNE VV2; E0(the letter O)R @4;STA L;
     DEY;BNE VV1;RTS;]
40   P.&6;G.b
100a IF?#B001&#80+0 A=A-3*Y
120  IF#B002&#40+0 B=B-3*X
130  A=A+Y;B=B+X
140  ?#80=3;LINK VV0
150  F=F+P;G=G+Q
155  IF R.%9=8 F=F+R.%4
160  IF(A<1)/(A>127)  Y = - Y
170  IF(F<1)/(F>127)  P = - P
180  IF(B<1)/(B>190) X=-X
190  IF(G<1)/(G>190) Q=-Q
```

```
200  PLOT15,C,D;C=A;D=B;PLOT13,A,B
210  PLOT15,H,J;H=F;J=G;PLOT13,F,G
220  IFABS(A-F)+ABS(B-G)<5 G,c
230  G,a
240b A=1;B=1;Y=2;X=2
250  CLEAR3
260 F= A,R%128;G=A,R,%b0;P=1+A,R,%2;Q+1+A,R,%2
270  G,a
280c MOVEA,B
290  PLOT5,A,B
300  A=A+R,%3
310  B=B+R,%3
320  ?#80=A,R,%30+8;LINK VV0
330  G,290
340  END
```

Another version of this program, written this time in graph graphic mode 1, is in the section near the back of the book.

The spaceships sound more like Sopwirth Camels than interstellar trains on the Central Line, and they sound like demented garden gnomes when they hit and proceed to fill the universe with a chaotic design of their own choosing. If you want to make it more difficult, change the 5 in line 220 to 2, or even to zero.



PEEK AND POKE

Most microcomputers (those that use BASIC) have the PEEK and POKE functions as part of their language. PEEK and POKE are surrounded by an aura of mystique and often seem far more frightening than they are. This section will by no means exhaust the wonders of PEEK and POKE, but it should tell you enough to start thinking about, and using, these two useful commands.

Before you can fully understand the mechanism of PEEKing and POKEing, and get the best use from them, you need a little bit of information about ROM and RAM.

The Atom's memory can be likened to a large box filled with little test tubes, each test tube containing a single number between zero and 255. These test tubes (or addresses/memory locations) and labelled consecutively from zero upwards. The Atom needs two boxes of test tubes -- one for the ROM and the other for the RAM.

The ROM (Read-Only Memory) is a box full of test tubes with the corks jammed on so tightly that you cannot get inside the test tubes to change the numbers they hold. You can only look at the test tube and see what number is stored there. This is called PEEKing. You can PEEK a specific address (look into a specific test tube) and find out what number is stored there. You cannot change the number.

The other box of test tubes is the RAM (Random Access Memory). These test tubes are labelled in exactly the same way as the ROM, but the RAM tubes do not have corks in them. So as well as looking at the number that is stored at that address, you can change that number. This is called POKEing. You can POKE a specific address and change the number which is stored there.

While it is perfectly safe to PEEK any address on the Atom, you should be careful when POKEing. You will not

do any lasting damage if you POKE numbers into certain areas of the ROM, but you may lose the stored text, or have to reset the Atom with BREAK.

Each address can hold a number between zero and 255 (00000000 to 11111111 in binary). Computers use binary code to store information (and there is more about this in the section on Assembler), so the largest number that can be held in an address in binary code is 11111111, which is eight bits (Binary Digits) long. This unit, of eight bits, is often called a byte. A computer with one kilobyte (1K) or memory (1024 bytes) has 1024 addresses.

You've probably noticed that the Atom BASIC does not contain the functions PEEK and POKE *per se*. When Atom BASIC was being written, it was decided not to incorporate PEEK and POKE as two different functions, but to use the query (?) and let the context of the use determine whether it was meant to mean PEEK or POKE. For example:

```
PRINT ?#80
```

The 'query' is used to represent PEEK here (and the line means print the value stored at HEX address 80). If you wanted to change the number at this address, you'd need to use a line like:

```
?#80=128
```

This time the query is used to represent POKE. POKE the address #80 with 128 (that is, change the number in address #80 to 128). The use of a query instead of PEEK and POKE is an ingenious (and probably unique) method. As well as being memory-saving, it is also quicker to input a ? than PEEK or POKE every time.

It was mentioned earlier that the context of the use of the query determined whether it indicated PEEK or POKE.

Here are a few examples of typical occasions where PEEK and POKE are used:

PEEK:

1/ PRINT ?#80 (print number in address #80)

2/ IF ?#80=128 GOTO 10 (if the number in address #80 equals 128 then go goto line 10, if not, pass onto the next line of the program)

3/ A=?#80 (let the variable A equal the number in address #80)

POKE:

?#80 = 128 (change the number in address #80 to 128)

You can see that PEEK (2) and POKE (1) are very similar, and it is only the prefixing by the conditional IF that determines whether *?#80=128* is a POKE or a PEEK.

Now that we've discussed the theory of PEEKing and POKEing, let's get down to the computer and try a few examples. We'll look at PEEKing first. Address #E1 is where the value of the cursor (that annoying white blob on the screen) is stored. Try typing in:

```
PRINT ?#E1
```

If all has gone well, when you pressed RETURN the Atom PEEKed (looked at) the value stored at #E1 and printed that value on the screen. It should have printed out 128, the code for an inverse space. Now try this one:

```
PRINT ?#1
```

You should get the answer 0. Now try it again in a

slightly different form:

```
10    PRINT ?#1
```

As we have now given it a line number, you will have to used the command RUN. You will now get the answer 10. Try running it with different line numbers which are less than 256. You can see that the answer corresponds to the line number, because address#1(and#2 for numbers over 255) contains the BASIC line number of the line being executed. Try looking at other addresses using the PEEK command. If you want to look at a block of addresses you could use the following program:

```
10    A=0
20    PRINT?A
30    A=A+1
40    GOTO20
```

As you can see, this is a very unwieldy way of doing things. Atom BASIC makes it very much simpler:

```
FOR A=0TO100;PRINT A?#0;NEXT
```

In *PRINT A?#0*, A is automatically added to the address to be PEEKed. As A is the controlling variable of the loop, A is incremented by one each time the loop is passed through. Therefore, as A is added to *#0*, the address to be PEEKed is incremented by one each time.

Now that we've seen a bit about what PEEK can do, let's turn our attention to POKE. If you remember, we used the example of *PRINT ?#E1* while discussing PEEKing. We PEEKed the address *#E1* and found it contained 128, the code for an inverse space. We can change this value, using the POKE command, to dramatic effect. Try this:

```
?#E1=0
```

You should find that the cursor has disappeared. We have, in effect, turned the cursor off. To turn if back on simply POKE address *#E1* with 128:

```
?#E1=128
```

Try POKEing this address with different values up to 255 and see what you get. Turning the cursor off can be extremely useful, especially when you want to smarten up the display. But don't forget to turn it on again if you want to edit the listing or you won't know where you are.

Although PEEKing and POKEing specific addresses in RAM (and PEEKing addresses in the ROM) can be very useful, the most practical use of these two functions is probably the direct PEEKing and POKEing of the TV screen. This is how you can make things move very quickly on the screen, without reprinting the entire screen every time.

The Atom is a 'memory-mapped' computer. This means that there is a specific portion of memory allocated for specific tasks (see section 25.6 of the Atom manual for a full diagram). The contents of addresses #8000 to #81FF have been set aside to hold the screen information - at one character cell per address (or byte). This is for the character mode only. If you use any of the graphics modes (except for CLEAR 0) you will use another specific block of memory. The use of the other graphic modes is discussed in the *Mastering the Graphics* section of this book. The address of the top left-hand cell of the screen is *#8000*. Try typing in:

```
?#8000=1
```

The letter A should appear in the top left-hand corner of the screen.

```
?#8010=1
```

The letter A should now appear in the middle of the top line of the screen, as *#10* equals 16 in decimal, which is half of 32, the number of characters along a line.

It is the value you stored in address *#8010* that determines the character displayed on the screen, and there are 256 different characters which can be displayed. These are the characters with codes between 0 and 255. The complete character code can be shown on the screen by running this short routine:

```
FOR A= 0 TO 222;A?#8000=A;N.
```

As we saw when we looked at PEEKing, a consecutive block of addresses can be PEEKed by adding the controlling variable of the loop to the address being PEEKed. We can do the same with POKEing.

As well as POKEing directly onto the screen, we can PEEK the addresses to find out what is in a particular position on the screen. We can see which character is where on the screen, but the Atom can only tell what is on the screen by PEEKing the addresses. Try entering, and running, this:

```
?#8010=1;PRINT ?#8010
```

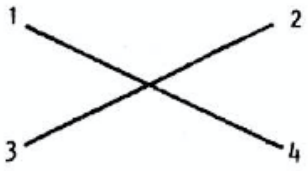
The first half of the line POKEs the character 1 (the letter A) into the middle of the top line of the s screen. The second half PEEKs address *#8010* and prints the value stored there. It is one in this case because we have just POKEd one into it. If you PEEK the next address (or any other blank square on the screen) you

will get the value 32, the value for a space.

With these two functions we can put any character into any square on the screen instantly, and at any time we can tell which character occupies which specific cell on the screen. However, you should remember than once the characters are POKEd onto the screen, they are subject to scrolling. If the screen scrools up, the characters will move from the cells where you put them.

The best way to appreciate PEEKing and POKEing the TV is to see it in action. The following two programs have been broken down and annotated step by fascinating (!) step so you can see exactly what is going on. The first program is called MALACHI and uses a chess board as the grid for the game.

In the game of MALACHI, your piece appears as an asterisk on the right-hand side of the board. The Atom's pieces are plings (exclamation marks) on the four left-hand squares of the board. Your piece can move in four directions:



The object of MALACHI is to elminate all four of the Atom's plings in the shortest number of moves. To elminate a pling, you have to land on it. But the Atom has a couple of nasty tricks up its sleeve. At random intervals throughout the game, the Atom will play its Defender pieces (inverse greater than signs) on the next row out from the plings. You have to clear the row of the Defenders to win the game. The game automatically stops when this row is cleared, even if some plings have survived. The number of surviving plings is deducted

from your score. Not the most intellectually stimulating of games, but a lot of fun (and frustrating as well) to play.

Here's how we went about writing MALACHI.

```
10    P.£12;@=0;C=0;D=150
50    P."(6 SPACES  10 ASTERISKS)"'
60    F.E=1TO4;P."(6 SPACES)"£#2A" "£128" "£128" "
      £128" "£128,£#2A
```

NOTE: There is a single space within the quote marks in line 60 and line 70

```
70    P."(6 SPACES)"£#2A,£128" "£128" "£128" "£#2A';N.
80    P."(6 SPACES  10 ASTERISKS)"''
```

Line 10 clears the screen, sets the numeric field to zero and assigns the variables. C is the number of goes you take and D is the base score from which C is subtracted to give you your score.

Lines 50 to 80 print out the board, using inverse spaces and asterisks. The asterisks are just to give the board a border.

```
110   F=71;F.G=1TO4;F?#8000=#A1;F=F+64;N.
```

Line 110 POKEs the Atom's plings into their positions on the board. The variable F is set to 71 which, when added to *#8000*, gives us the address of the first white left-hand side square of the board. This address is poked with the code for a pling. F is then incremented by 64 to give us the address of the cell in the same position along the line, but two lines lower. That is, the next white square down the left of the board. The loop from one to four POKEs all four of the Atom's plings into place.

```
160   H=174;H?#8000=#AA
```

The Atom now POKEs your piece into place. The variable H is set to 174, which when added to *#8000* gives us the address of the third white square down the the right side of the board. Code #AA is the code for an asterisk.

```
180   P."WHICH DIRECTION WILL YOU MOVE";IN.A;P.$11
200   C=C+1;F=71+((A.R.%4+1)*64)
```

The board is set up ready to play, so the Atom asks you for your move. The command P. $11 at the end of line 180 is very important at this point. As we mentioned earlier, if the screen scrolls upwards, the characters move from the cells where you put them. So to stop this happening we use $11, which moves the cursor back up one line.

In line 200 the number of goes you have had is increased by one. The variable F is set to correspond to one of the four squares with plings on when added to *#8000*. This is to determine which of the four base squares has the chance of sending out a Defender piece into the next row.

```
250   K=A.R.%4+1;IF K>2 G.310
270   IF K = 1   F=F+33
280   IF K = 2   F=F-31
```

The variable K is set to a random number between one and four. If the random number is greater than two, the Atom does not send out a Defender piece but skips the following routine and goes to line 310. However, if the number is one or two, the Atom decides where to send the Defender piece.

```
290   IF F?#8000=74 G.320
300   F?#8000=#BE
```

Once it has made its choice, it checks to see if the square it has chosen is vacant. In other words, it

PEEKs the address F#8000. If address F#8000 contains the vale 74 (the ASCII code for the asterisk), the Atom knows that it has chosen a square off the board and forfeits its move. If the PEEKed value of F#8000 does not equal 74, it means that it has chosen a valid square on the board. Line 300 then proceeds to POKE the value #BE into that address.

```
310   H?#8000=127
320   IF A=1 H=H-33
330   IF A=2 H=H+31
340   IF A=3 H=H+33
350   IF A=4 H=H-31
```

The variable H, you may remember, is assigned to the number which, when added to #8000, gives the address of your playing piece. The square your piece is currently on is made blank (POKEd with 127). H is then reset depending on the direction in which you decided to move.

```
360   IF H?#8000=74 G.160
370   H?#8000=#AA; IF C < 5 G.180
```

If you decide to move off the board, line 360 will stop you and send you back to the start. In the same way that the Atom checks if it is moving off the board, it checks to see if you are trying to move off the board. Assuming that you have moved correctly, the Atom POKEs your piece (the inverted asterisk) into your chosen square.

The IF condition at the end of line 370 stops the Atom going into the 'win-check' routine for the first five moves, because it is possible to have the Atom's second rown blank before the first Defender piece is positioned.

```
400   IF ?#8028<>190;IF?#8068<>190;IF?#80A8<>
      190;IF?#80EA<>190  G.a
```

```
410   G.180
```

Line 400 is the 'win-check' routine. It takes advantage of the way Atom Logic is set up. The Atom will PEEK each of its second row squares in turn, and onyl if the value found is not equal to 190 (the value of the greater than character) will it pass along the line to check the next square. If, and only if, all four squares contain values not equal to 190 will the Atom obey the command G.a. If at least one of the squares contains a Defender, the Atom will be sent to line 180 to continue the game.

```
415 a T=71;R=0
420   IFT?#8000=#A1  C=C+10
425   T=T+64;R=R+1;IF R < 4. G.420
430   F.W=1TO60;WAIT;N.;P.'
440   P."END OF GAME"''D-C"  POINTS";END
```

Using lines 420 and 425, the Atom checks to see if you have taken all its plings.

If the PEEKed value of an address equals *#A1*, it means that a pling is still present, and 10 is added to your score. After a short wait, your final score is printed as 150 minus the number of goes you took, plus any for forfeits for leaving any plings behind. The higher the score, the better.

You should, by running this game and closely following each step as it occurs, now appreciate how PEEKing and POKEing directly into the screen work. And you should also appreciate the versatiltty they give you for your game screen displays. POKEing to the screen is, to all intents and purposes, instantaneous. At no time during the game of MALACHI does the screen have to be cleared, and the board reprinted.

As a contrast to the game of MALACHI, we've included a rather amusing little game called TARTAN ARMY. We're not going to analyse this game as deeply as we did the previous one, as there is no point in explaining the same things twice. But to play the game, you need to know the rules.

You are a lone policeman (an inverted asterisk) on duty outside Wembley. You have several policevans with you (the letter O's). The crowd pours out after the game, and a rampage begins. How many hooligans can you arrest (by landing on the same square) before they overturn all of your policevans...or do you a personal injury (which is much more painful than an impersonal injury). You can move anywhere on the white area of the screen by entering a number (1 is one square to the right; -32 is one square above; 64 is two squares below; and so on).

While the game itself does not achieve very much, it's good fun to play and is a good demonstration of PEEKing and POKEing the TV screen.

```
5     P.£12;@=0
10    F.F=1TO320;P.£128;N.
```

Lines 5 and 10 clear the screen and print out the white 'patrol area'.

```
20    T=0;F.F=0TO319;K=A.R.%25;IF K = 1 F?#8000=#8F
30    IF K<>1  F?#8000=127
40    N.
120   X=A.R.%320;X?#8000=#AA
```

Line 20 decides whereato putthe police vans (character #8F) and line 120 decides where you policeman (character #AA) starts his beat.

```
130   M=0
```

```
150   F.F=1TO320;IF F?#8000=#8F M=M+1
160   N.
240   P.'"HOOLIGANS ARRESTED: "T"     "'"POLICE
          VANS LEFT: "M"     "'
250   P."(6 SPACES)"'';P.$11;P.$11
260   IF M   1 G. c
370   IN.Z;W=X+Z
```

Line 150 checks how many police vans are left. Line 240 prints out the current situation. You get one point for each hooligan you arrest. The instruction to move the cursor up two lines (P.$11;P.$11) are very important. If you don't include these, the scrolling caused by the printing would corrupt the screen positi positions and ruin the game.

Line 260 checks to see if you have any vans left.

```
390   IF W?#8000=143  G.a
400   IF W?#8000=190  G.b
405   IF W?#8000  127 P.$11;G.370
```

These three lines use the PEEK function to check what value was stored in the square which you've just moved to. If it was 143 (inverse letter O) you go to subroutine a where you lose lose five points for upsetting your own police van. However, if it is a 190 (an inverse greater than) you go to subroutine b where you gain a point for successfully arresting a hooligan. If the value was not one of these, and is not a 127, you must have moved off the patrol area, and so your move is rejected.

```
406   X?#8000=127
410   W?#8000=170;X=W
```

The computer now POKEs your policeman into his new posltion...

```
440   F.F=1TOA.R.%9+1;Q=A.R.%320;IF Q?#8000=170 G.c
```

```
450   Q?#8000=#BE;N.
470   F.F=1TO4;P.&11;N.;G.130
```

...but the Atom gets its own back by increasing the size of the crowd. If the computer wants to POKE a hooligan onto the square your policeman occupies, subroutine c is called-

```
500a  T=T-5;G.406
510b  T=T+1;G.406
520c  P.&7;F.F=1TO100;X?#8000=#66;WAIT;X?#8000=#59;
      WAIT;N.
530   X?#8000=#BA;F.F=1TO4;P.&11;N.;P.''''';F.F=1TO
      4;P.&11;N.
540   P."YOU HAVE HAD IT"';P.&11
550   P."YOU HAVE HAD IT "';P.&11;G.540
```

Subroutine c(lines 520 to 550) deserve some mention. P.$7 causes the Atom to give a short beep. The rest of the line rapidly POKEs alternating graphical characters into the square where your hapless policeman was standing. He finally ends up as a colon. Lines 550 and 540 alternatively print out the words "YOU HAVE HAD IT" in ordinary then inverse letters. Moving the cursor up each time gives a flashing effect and keeps the dislay intact.

These two programs go some way to showing you waht can be achieved with PEEKing and POKEing. Many other programs in this book make use of these functions. One program you might like to try is *TRAFFIC JAM* which is at the end of the section on keyboard scans. You should be able to work out how this game works using the information in this section.

ARRAYS

Arrays puzzle many newcomers to BASIC. You are lead to believe arrays are specialised functions when in fact they're just another set of variables. Atom BASIC allows you 26 variables (A to Z) and for most programs you write these should suffice. But there are occasions where you'll need more than 26 variables (such as The Forever War later in this book) or you want a greater degree of flexibility in comparing variables. Arrays become extremely useful here.

A lot of confusion with arrays is caused by the DIM statement. To print something you use PRINT, to input something you use INPUT, but to use an array you must use DIM. DIM is a contraction of the word dimension and this statement is used to set up an array.

Let's just check out what an array is, before proceeding. An array is represented by the array name (identifier) and the subscript (for example in the array element AA(5) the 5 is the subscript). Note that Atom BASIC uses paired letters (from AA, through BB all the way to ZZ). AA(0) represents the first element of the array AA. Imagine a long row of matchboxes stuck side to side. Each matchbox is one element in the array. So AA(0) is the first element/matchbox in the array with the name AA. The second matchbox is AA(1) and so on.

As we said before, you need to dimension an array before you can use it. By dimensioning, you tell the Atom how many elements you want to use in an array. Entering DIM AA(20) tells the Atom you want it to reserve space for 21 elements in the array of the name AA. The computer sets aside four bytes for each element of the array so you can use numbers between approximately 2000 million and minus 2000 million.

Arrays are often used to create elements to take the place of ordinary variables. Once an array has been dimesnioned you can use any element of that array as you would a simple variable:

```
A = 10  .....or
AA(3) = 10
```

In both of these cases, the computer files the value 10 in a specific area of its memory. If you now type in the line 10 DIM AA(6) the Atom will allocate space fo for the array AA starting at the first free address memory location after the program text. If you were to subsequently dimension an array named BB, the space for this second array would be allocated starting from the first free address after array AA. If you add the line 20 AA(5) = 10 the Atom will store the value 10 in the sixth element of array AA:

AA(0)	AA(1)	AA(2)	AA(3)	AA(4)	AA(5)	AA(6)
-	-	-	-	-	10	-

Until they're allocated a value, the other elements of the array contain unspecified random numbers. Now, add another line: 30 P. AA(5). This will PRINT out the value stored in AA(5) which we have just set to 10. Once a value has been stored in an array it is very easy to change it. A(5) = 100 will remove the value already stored (10) and replace it with the new value (100). The subscripts of the arrays, as well as being numbers, can also be variables. You could, for example, come across AA(X) in a program listing. Try this program:

```
10    DIM AA(5)
20    INPUT X
30    AA(X) = 100
```

The variable name AA is dimension to allow for six

elements. Line 20 asks you to enter a number. If you enter a number between 0 and 5 the Atom will store the value 100 in the particular element of the array that you specified. So, if you entered the number 4, the computer would store the value 100 in the fourth element of the array:

AA(0)	AA(1)	AA(2)	AA(3)	AA(4)	AA(5)
-	-	-	-	100	-

The ability to use a variable as an array subscript is very useful when entering information in FOR-NEXT loops. For example:

```
10    DIM AA(6)
20    FOR F=0TO6
30    INPUT X
40    AA(F)=X
50    NEXT
```

This routine uses the controlling variable of the FOR-NEXT loop as the array subscript. The first time through the loop F will equal zero. If you enter the value 20 (line 30), line 40 will store that value in element F (0 in this case) of the array. The second time through the loop, F will equal one. So the next number you enter will be stored in element F, where F equals one. Add the following lines and then run the program:

```
15    P."ENTER 7 NUMBERS"'
60    FOR F=0TO6
70    P. AA(F)'
80    NEXT
90    END
```

The first half of the program stores the values that you enter into the array, and the second half prints them out. We've now seen what arrays are. Let's look at a proper program to see how they work.

```
10    DIM AA(6)
20    FOR B=1TO10;P."ENTER A NUMBER";INPUT X;
      AA(B)=X;N.
```

The first two lines of the program are much the same as in the previous program. A FOR-NEXT loop (line 20) accepts a series of numbers and stores them in ascending elements of the array (controlled by the variable B).

```
30    FOR B=1TO9;FOR C=B+1 TO 10;IFAA(B) < AA(C)
      G. 50
40    D=AA(B);AA(B)=AA(C);AA(C)=D
50    NEXT B;NEXT C
```

Lines 30 to 50 contain a loop within a loop. Adjacent elements of the array are compared (line 30). AA(C) is the next higher element to AA(B), so if the value in AA(B) is less than the value in AA(C) the program moves on to compare the higher of the previous two elements with the next higher element. If, however, the value in AA(B) is not less than the value in AA(C), line 40 swaps the values so the value in AA(B) is lower.

```
60    P. ''''
70    FOR B=1TO10;P.AA(B)-N.;END
```

The program has now sorted out the numbers you entered into ascending numerical order, and line 70 prints these out. This is an example of a 'bubble sort', where the 'lighter' elements move to the top of the array. To print out the numbers in a descending sequence, simply change line 70 to:

```
70    FORB=10TO1 STEP -1;P.AA9B0;N.;END
```

There are five important points about arrays that should be remembered:

.Arrays must be dimensioned by using the DIM

statement. If you dimension an array with 20 (DIM AA(20)), you can still (if you really want to) enter values into the array with a subscript greater than 20. The Atom will accept them, but the practice is most unwise because the computer has only set aside memory space for subscripts up to 20. Any subscripts above the dimensioned value will possibly corrupt other variables stored in the Atom's memory

.Each array has an many elements for storage of values as the number of the subscript plus one. That is, the subscript starts at zero

.The subscript can be another variable (A,T, Z or whatever) so long as the variable has previously been given a number

.The subscript variable can be the controlling variable of a FOR-NEXT loop, as we saw in the previous program. However, an arrayed variable cannot be the controlling variable of a FOR-NEXT loop. You cannot say, for example: FOR AA(2) = 1 TO 7. You must use a simple variable in this case

.You cannot input a value directly into a dimensioned variable. INPUT AA(X) would give you an error code. Here again, you have to use simple variables as we did in the previous programs

While the point was made that you should dimension enough elements for the array you need, it is a waste of time (and memory) to dimension too many. There is no point in dimensioning an array with 100 if you only need 10 elements. When you save a program which contains DIM statements in it, the Atom will save each element of the array separately. This can add several unnecessary seconds to the loading and saving times of programs.

The standard Atom arrays are one-dimensional. They have just one subscript. Some BASICs allow multi-

dimensional arrays (e.g. A(2,4,6)), but it is not possible on the Atom. However, there is a way of representing these matrices by simple manipulation of the one-dimensional arrays.

Imagine a grid of five squares by five squares, with the number 8 at the co-ordinates 3,2:

```
  0 1 2 3 4 5
1 . . . . .
2 . . . . .
3 . 8 . . .
4 . . . . .
5 . . . . .
```

There are 25 elements in this array (5 X 5). A two-dimensional array would represent this as A(3,2), but with the Atom's one-dimensional array we have to count the squares (top to bottom, left to right). We find that it is the 18th element of the array AA(17). You don't have to count the squares every time as there is a simple formula to do it fo you:

Let X equal the number of squares down the grid
(0 to X - 1)
Let Y equal the number of squares across the grid
(0 to Y - 1)
So, to dimension the number of elements you need, you:
*DIM AA(X*Y - 1)*

For example: DIM AA(5*5 - 1) or DIM AA(24)

To find the position of any element of the array on the grid, taking our example again:

Let A equal the number of squares that the element is down the grid (3 in this case)
Let B equal the number of squares that the element is across the grid (2)

So we find that the value 8 is in element:

AA(A * Y + B)

or AA(3 * 5 + 2)

or AA(17) as we found earlier by actually counting the squares on the grid.

The following program will help you to understand the use of a matrix with a one-dimensional array.

```
5   INPUT X,Y
10  DIM AA(X*Y - 1);Q=0
11  FOR F= 0 TO X*Y-1;AA(F)=142;N.
12  INPUT A,B
13  AA(A*Y+B(=132
15  FOR G=1TO Y
20  FOR F=1TO X;P.$AA(Q);Q=Q+1;N.;P.'
30  N.;G.5
```

Enter the values for X and Y to determine the size of the grid. Then enter the co-ordinates for a specific element (A is down, B is across). The computer will then print out the grid highlighting the element you have chosen.

As well as the standard arrays, the Atom has a couple of variations called byte vectors and word vectors.

Byte vectors are very similar to the standard arrays, but in this case, only one byte is allocated per element, instead of four. These are useful when memory is tight and you do not need numbers greater than 255 - the highest number that can be stored in one byte.

Instead of using paired letters as the array name, byte vectors use the simple variable names. You have to dimension them in the same way as you do standard variables: DIM A(6). However, if you decide to use A as a byte vector array, you cannot use it as

a simple variable as well. This would confuse the poor little Atom and your values for the variables would be corrupted. When you wish to use a byte vector, you write vector elements as:

```
A?0, A?1,A?2.........A?27
```

intead of AA(0), AA(2) etc as we saw with the standard arrays. The value after the question mark is still called the subscript and the letter before it is still called the name/identifier. Byte vectors obey the same five main rules as the standard arrays.

The program SWAPPO at the end of this section uses standard arrays, although it is possible to change them for byte vectors. We have left the program so it uses standard arrays so you can get some familiarity with byte vectors by converting the program if you are interested. Now that you've mastered byte vectors, we come to word vectors.

Words vectors are very similar to byte vectors, except that a pling (!) is used instead of the question mark. Four bytes are allocated per element in word vectors. So for the word vector W with six elements, we have:

```
W!0   W!4    W!8    W!12   W!16   W!20
```

The element subscripts are the numbers to the right of the pling - in steps of four. Word vectors also obey the standard array rules. They are mentioned here for completeness, although for most problems involving arrays, the standard ones (AA to ZZ) are more convenient.

The following program, called SWAPPO, is an easy game to play, but it serves to illustrate the versatility of arrays in place of simple variables. The standard arrays can be converted to byte vectors in this program with very little alteration.

```
10  P.$12;P."SWAPPO"';P.';@=0;?#E1=0
15  DIM AA(20);X=1
20  B=9;AA(1)=A.R.%9+1
```

The first three lines of the program clear the screen, switch the cursor off and initialise the variables, including dimensioning the array AA.

```
30  F.C=2TOB
40  AA(C)=A.R.%9+1;X=1
45  FOR D=1 TO C-1
50  IF AA(C)=AA(D) X=3
55  N.D
56  IF X=3 G.40
60  X=1;N.
```

Lines 30 to 60 set up the random starting sequence of the nine numbers. Line 50 compares two adjacent elements of the array. If they are the same, the computer is sent back to line 40 to choose a different number.

```
70  P."PRESS RETURN TO FIND OUT YOUR"'"STARTING
    SEQUENCE"'
75  LINK #FFE3;E=0;P.';P.$12;GOS.200
```

The Atom is sent to the subroutine at line 200 after it has run through the instructions. We'll skip over the next few lines of the program to enter the subroutine.

```
200 ?#E1=0
210 F. C=1TO B;P.AA(C)"   ";N.;P.''
```

The first part of the subroutine turns the cursor off again (p.$12 switches it on) and the next line prints out the numbers in the order that they are stored in the array.

```
212 ?#8091=32
215 F.Y=1TO10
```

```
220 F.W=0TO15;W?#8080=W?#8080+128;N.
222 F.W=0TO15;W?#8080=W?#8080-128;N.;N.
230 R.
```

The second part of the subroutine causes the flashing on the words HOW MANY TO SWAP by alternately POKEing the squares with the normanl and inverse letters.

```
80  P.'''"HOW MANY TO SWAP";IN.F
90  IF F=0 G.170
100 IF F<= B G.120
110 P."I CAN ONLY SWAP "B';G.80
```

Lines 80 to 110 accept your input of how large a block of numbers you want to change. For example, if the n numbers were arranged:

6 4 1 2 9 8 3 7 5

and you chose to swap five, the five numbers to the left would swap places:

9 2 1 4 6 8 3 7 5

The idea of the game is to arrange the numbers in sequence from left to right in as few goes as possible.

```
120 E=E+1;F.C=1TO F/2
130 G =AA(C);AA(C)= AA(F-C+1);AA(F-C+1)=G
140 N.;P.$30;@=0;GOS.200
```

These three lines swap the numbers around in the block of numbers you chose. As you've probably noticed, there are similarities with parts of the bubble sort given a little earlier.

```
150 C=1
151 IF AA(C)<> C G.80
152 C=C+1;IF C<> 9 G.151
160 P."YOU SUCCEEDED IN "E" MOVES"''
170 P."DO YOU WANT TO TRY AGAIN"'"YES (1),
    NO (2)"'
180 IN.N;IF N=1 G.10
190 P."THANKS FOR THE GAME"';?#E1=128;G.190
```

The last few lines of the program check to see if you have succeeded in re-arranging the number sequence, and if you have, whether you wish to play again.

MASTERING THE STRING FUNCTIONS

Strings are not the easiest things to manipulate in Atom BASIC. However, all the common BASIC string functions (MID$, RIGHT$ etc) can be emulated fairly easily, if not elegantly.

Strings are a sequence of characters (letters, numbers or graphical symbols) with each character being represented by a code value between 0 and 255. When you print a string a characters on the screen, you enclose them with quote marks:
PRINT "ACORN ATOM" appears as ACORN ATOM

If you want to manipulate the strings at all, you must use string variables. The string variables ($A to $Z) must be dimensioned like numerical arrays. In fact, they can be likened to a string array where each character occupies one element of the array.

```
DIM A(10)
```

will allocate space for a string of up to 10 characters (including spaces and punctuation). In the same way, you assign values to numerical array variables, you can assign strings to string variables:

```
$A="ACORN ATOM"
```

DIM A(10) has reserved 11 addresses in the memory immediately above the program text. Therefore, $A is stored as:

```
    0 1 2 3 4 5 6 7 8 9 10
A = A C O R N   A T O M  -
```

Once the string has been assigned to a string variable

it can be printed at any time:

```
PRINT $A
```

will appear as

```
ACORN ATOM
```

Although you can't input values directly into arrays, you can enter strings directly into string variables, so long as the string variable has first been dimensioned. The maximum length of string that can be typed into an input statement is 64 characters.

If you dimension a second string, it's possible to assign the value of the first string to the second:

```
10  DIM A(10), B(10)
20  $A="CHOPPER"
30  $B=$A
40  PRINT $A
50  PRINT $B;END
```

This program would print out the word Chopper twice, once for each string. You can also compare strings. Delete line 50 and change 30 and 40 to:

```
30  $B="CHOPPER"
40  IF $A=$B PRINT "THEY ARE THE SAME"
```

The computer will print out, not surprisingly, *THEY ARE THE SAME.*

Atom BASIC has several other string functions which help with the manipulation of strings. One of these is the length of a string (LEN). This function tells you how many characters (including spaces and punctuation) there are in a particular string:

```
10  DIM A(10)
20  $A="ACORN ATOM"
30  PRINT LEN(A)
```

This will print the answer 10 because the string "ACORN ATOM" contains 10 characters.

Another useful function is CH. This gives the ASCII code of the first character of any particular string.

```
CH "A"
```

will give you the value 65 the ASCII code for A.

The character extraction functions are probably the most useful string functions. The simplest of these is the single character extraction. Let's look again at the representation of string A:

```
    0 1 2 3 4 5 6 7 8 9 10
A = A C O R N   A T O M -
```

We can find the value of any character in a string by using the statement A?N where N is a number between zero and the highest element of the string array (10 in this instance).

```
A?3 = R
A?4 = N
A?9 = M
```

This next program demonstrates the use of the query operator by inverting a string after you've entered it:

```
10  DIM A(10)
20  INPUT $B
30  FOR F=0 TO LEN(B)-1
40  B?F=B?F+32
50  NEXT
```

```
60  PRINT $B
70  G. 10
```

If you want to extract more of a string than a single character, there are short routines for right-string, left-string and mid-string extraction.

Here's how we'd go about extracting the word ATOM from $A. First of all, another string has to be dimensioned to accept the part of $A we wish to extract.

```
10  DIM A(10),B(10)
20  $A = "ACORN ATOM"
```

We now want to extract the last four characters. In other words, we don't want the first six characters of the string. So we have to say:

```
30  $B=$A+6
```

What we are really saying is that string B is equal to everything in string A except the first six characters. If you add line 40...

```
40  P.$A',$B;END
```

...you'll see this will print out string A - ACORN ATOM - with string B - ATOM - below it. The original string has not been destroyed. If however we did not need to use string A again, we wouldn't have to dimension a second string. Then, we could change line 30 to...

```
30  $A=$A+6
```

...which re-assigns the new string to string A. This process is called right-string extraction. Left-string extraction is, of course, exactly the opposite.

```
10  DIM A(10)
20  $A="ACORN ATOM"
```

If we wish to extract the first five characters we have to say:

```
30  $A+5=""
40  PRINT $A;END
```

This will print the word ACORN, the first five characters of string A.

By combining the two above techniques we can get mid-string extraction. A good rule to follow here is to do any left-string extraction before tackling the right-string extraction.

If we wanted to remove the first and last two characters of the string, we'd proceed as follows:

```
10  DIM A(10)
20  $A="ACORN ATOM"
30  $A+8=""
40  $A=$A+2
50  P.$A;END
```

Line 40 carries out the left-string extraction, removing the last two characters from the string. Line 50 then does the right-string extraction to remove the first two characters. This leaves the following to be printed:

```
    ORN AT
```

The following program demonstrates the extraction functions functions at work:

```
10  P.'''''';DIM A(64)
20  P."ENTER STRING";IN. $A
25  L=LEN(A)
```

```
30  K=A.R.%2+1;IF K = 1 G.100
40  IF K = 2 G.200
100 $A=$A+A.R.%L;P.$A;G.10
200 $A+A.R.%L;P.$A;G.10
```

This program randomly chooses whether to perform a left-string or right-string extraction. Once it has been chosen, the appropriate routine (100 or 200) executes the extraction on a random length of the string.

One routine that is useful in many programs is stripping a string of its first character. There's a routine to do this in the next program.

```
10  DIM A(64)
20  P."ENTER YOUR STRING";IN.$A
30  F.F=1TO LEN(A)
40  P.$A '
50  $A=$A+1
60  NEXT
70  END
```

So, from this section you can see that while string manipulation is not as easy on the Atom as on some other computers, there are routines to emulate the most useful string functions.

THE ATOM IN EDUCATION

This section of the book is designed to:

i/Show the range of functions available on the extension (floating point ROM)

ii/Demonstrate how simple programs can be developed around commonly-used formulae

iii/Give ideas for problems to be set when using the Atom in a classroom situation.

Most of the programs require an Atom fitted with a floating point ROM, although there are a few which will run on the standard, integer ROM.

Firstly, we'll look at some ways the Atom can be used to print out tables.

MULTIPLICATION TABLES

```
5    @=0
10   IN."WHICH TIMES TABLE"A
20   F.B=1TO12
30   P.B" X "A" = "A*B'
40   N.;END
```

SINES

```
5    @=0
10   IN."LOWEST NUMBER"A
20   IN."HIGHEST"B
30   F.D=A TO B
40   %R=D*PI/180          (Changes radians to degrees)
50   P."SIN "D" IS ";FP.SIN%R'
60   N.;END
```

SQUARES

```
5    @=0
10   IN."LOWEST NUMBER"A
```

```
20    IN."HIGHEST"B
30    F.X= A TO B
40    P.X" SQUARED IS "X*X';N.;END
```

FLOATING POINT VERSION

```
5     @=0
10    FIN."LOWEST NUMBER"%A
15    %C=%A
20    FIN."HIGHEST"%B
25    %B=%B+%C
30    DO
40    FP.%A"  SQUARED = "%A*%A'
50    %A=%A+%C
60    UNTIL %A > %B
70    END
```

TEMPERATURE CONVERSION

```
10    IN."HIGHEST TEMP (F)"A
20    IN."LOWEST"B
30    IN."STEP"C
40    F.T=B TO A+C STEP C
50    M=5*(T-32)/9;K=M+273
60    P.T M K'
70    N.;END
```

MULTIPLICATION QUIZ

```
5     @=0
10    IN."DIFFICULTY"A
20    IN."NUMBER OF QUESTIONS"B
30    F.C=1 TO B;P.'
40    D=A*(A.R.%10+1)
50    E=A*(A.R.%10+1)
60    F=D*E
70    P."WHAT IS "D" TIMES "E
80    IN.G
90    IF G = F P."CORRECT";G.110
100   P."NO, THE ANSWER IS "F
110   N.;END
```

MONTHLY REPAYMENTS ON A LOAN

```
10   FIN."AMOUNT OF LOAN"%L
20   FIN."OVER HOW MANY YEARS"%Y
30   FIN."AT WHAT INTEREST RATE"%R
40   %A=%R/100
50   %P=%L*%A*(1+%A)^%Y/(12*((1+%A)^%Y-1))
60   FP."MONTHLY PAYMENT IS "%P'
70   FP."TOTAL REPAYMENT IS "%P*12*%Y'
80   END
```

(Note the use of the ^ for the upward arrow, raising to a power.)

COMPOUND INTEREST

This works out the balance if a sum of money (M) is deposited for a number of years (Y) at a certain rate of interest (R). You can enter fractions of years as, for example, 25 + 7/12 for twenty five years, seven months. An interesting exercise is to see how long $1000 must be left on deposit at 10 per cent to become a million dollars (the answer lies between 72.4765749 years 72.4765748 years, a surprisingly short time).

```
10   FIN."PRINCIPLE"%P
20   FIN."INTEREST RATE"%R;%R=1+%R/100
30   FIN."HOW MANY YEARS"%Y
40   FP."TOTAL ="%P*%R ^%Y;END
```

DATA SORT - INTEGER

```
10   DIMQQ100
20   IN."HOW MANY TO BE SORTED"A
30   F.B=1 TO A;IN."INPUT NUMBER "B'
40   IN.Z;QQ(B)=Z;N.
50   F.B=1TO (A-1); F=0
60   F.D=1TO(A-B);IFQQ(D)<=QQ(D+1) G.80
70   E=QQ(D);QQ(D)=QQ(D+1);QQ(D+1)=E
75   F=1
```

```
80   N.
90   IF F=0 G.110
100  N.
110  F.D=1TO A;P.D,QQ(D)';N;END
```

As listed here, the program will sort data into ascending order. To get it to sort into descending magnitude, simply change the greater than sign in line 60 to a less than. The next program is listed to sort floating point numbers into descending order, and uses a somewhat different approach. An examination of the differences is valuable. Note that you input an S when ready to sort.

DATA SORT - FLOATING POINT

```
5    FDIM%QQ100
10   Z=0;Y=10
20   S=RND
30   P."INPUT NUMBER "Z+1;FIN.%M;%QQ(Y)=%M
50   IF%QQ(Y)=S G.70
60   Z=Z+1;Y=Y+1;G.30
70   W=Z+8;T=Z+9
100  F. X = 10 TO W
120  F.N=(X+1) TO T
130  IF%QQ(X)> %QQ(N) G.170
140  %E=%QQ(X);%QQ(X)=%QQ(N);%QQ(N)=%E
170  N.;N.
190  F.N=1TO Z;P.N"  ";FP.%QQ(N+9)';N.
200  END
```

ARITHMETIC MEAN

```
10   @=0;%C=0
20   IN."HOW MANY WILL YOU ENTER"A
30   F.B=1 TO A
40   P."INPUT NUMBER "B;FIN.%D
50   P.(A-B)" TO GO"';%C=%C+%D
60   P.';P.';FP."TOTAL WAS "%C'
70   P.';P.';FP."MEAN IS "%C/A;END
```

PLOTTING DEMONSTRATIONS:

OVAL

```
5    CLEARU
10   F.A=1TO10U
20   %B=PI*A/50
30   %C=25*COS%B+25
40   %D=19@SIN%B+20
50   PLOT13,%C,%D
60   N.;END
```

PLOT/UNPLOT SINE CURVES

```
5    CLEARU;K=13
10   F.A=2TO6U
20   %B=A*PI/30
30   PLOTK,A,%(SIN%B*20+20)
40   N.;IFK=13 K=15; G.10
50   IF K=15 K=13; G.10
```

LOGARITHMS - BASE *e* BASE 10

```
10   FIN."NUMBER "%N
20   FP."NATURAL LOG "LOG %N'
30   FP."COMMON LOG "LOG%N*.4343
40   END
```

AREA OF A RECTANGLE

```
10   FIN."LENGTH"%A
20   FIN."BREADTH"%B
30.  %C=%A*%B
40   FP."THE AREA IS "%C;END
```

VOLUME OF A CUBE

```
10   FIN."LENGTH OF SIDE"%A
20   %V=%A ^ 3
30   FP."THE VOLUME IS "%V
40   END
```

PERIMETER OF A TRIANGLE

```
10   FIN,"SIDE 1"%A
20   FIN,"SIDE 2"%B
30   FIN,"SIDE 3"%C
40   %P=%A+%B+%C
50   FP,"THE PERIMETER IS "%P
60   END
```

LENGTH OF SIDE OF RIGHT-ANGLED TRIANGLE

This program is to find the length of a third side of a right-angled triangle, knowing the length of the other two sides, using the formula $X^2 = Y^2 + Z^2$.

```
10   FIN,"LENGTH SIDE 1"%Y
20   FIN,"SIDE 2"%Z
30   FP,"THIRD SIDE IS "SQR(%Y ^ 2 + %Z ^ 2)
40   END
```

AREA OF A TRIANGLE - TWO METHODS

Note the way the FINPUT statement is used in line 10.

```
10   FIN,"SIDES"%A,%B,%C
20   %D=(%A+%B+%C)/2
30   %E=SQR(%D*(%D-%A)*(%D-%B)*(%D-%C))
40   FP,"THE AREA IS "%E;END
```

```
10   FIN,"BASE"%B
20   FIN,"HEIGHT"%A
30   FP,"AREA IS "%B*%A/2;END
```

CIRCLE - CIRCUMFERENCE

```
10   FIN,"RADIUS"%R
20   FP,"CIRCUMFERENCE IS"2*PI*%R;END
```

CIRCLE - AREA

```
10   FIN."RADIUS"%R
20   FP."AREA IS "%R ^ 2*PI;END
```

CIRCLE - AREA OF A SECTOR

```
10   FIN."RADIUS"%R
20   FIN."ANGLE"%A
30   FP."AREA OF SECTOR IS "%A/360*PI*%R ^ 2;END
```

VOLUME OF A CONE

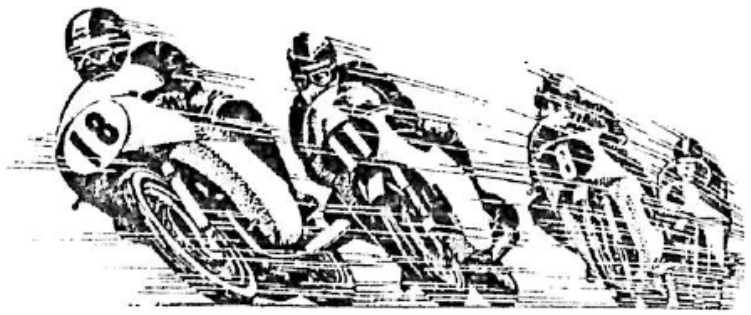
```
10   FIN."RADIUS"%R
20   FIN."HEIGHT"%H
30   FP."HEIGHT IS "PI*(%R ^ 2)*%H/3;END
```

SURFACE AREA OF A TANK

```
10   FIN."DIAMETER"%D
20   FIN."HEIGHT"%H
30   %A=((%D/2) ^ 2)*PI*2+PI*%D*%H
40   FP."AREA IS "%A;END
```

SPHERE - SURFACE AREA, VOLUME

```
10   FIN."RADIUS"%R
20   FP."SURFACE AREA "%R^ 2*4*PI
30   FP."VOLUME "4/3*PI*%R ^ 3;END
```



RANDOM NUMBERS

The random numbers generated by the RND function on your Acorn Atom are not truly random, although they are close enough for most purposes. The computer actually uses a long list of numbers, so long it appears random. The first program in this section draws a bar graph of the numbers generated by the RND function. RUN it a few times, then come back to this book.

```
20  F.A=1TO15
30  B=ABSRND%20+1
40  IF B<10 PRINT " "
50  PRINT B" "
60  F.C=1TOB
70  P."inverse x"
80  N.
90  P.'
100 N.
110 END
```

If you'd like to slow the printing down a little, add:

```
75  WAIT;WAIT
```

Another way of slowing such a program down is to use a loop. Try it with different numbers.

```
75  F.N=1TO75;N.
```

The RND function is used in many games to simulate the roll of a die. If it is working correctly, it should (like a proper die) produce each number between one and six an equal number of times. More realistically, the more you roll the die, the closer the distribution of each number between one and six should be the same.

The next program rolls a die as many times as you desire, and clocks up the totals for you as it runs.

When you run the program, a number prompt will appear. This is the number of times you want the die to be rolled. As the program progresses, you'll see a table being built up, which counts how many times each number appears.

```
10      IN.D
20      DIMAA6
30      F.K=1TO6;AA(K)=0;N.
40       @=3
50      F.C=1TOD
60      P.&12;?#E1=0
70      P." 1 2 3 4 5 6"'   (two spaces before the
80      K=ABSRND%6+1          first number, and two
90      AA(K)=AA(K)+1         between each one)
100     F.B=1TO6
110     P.AA(B);N.
120     WAIT
130     N.;END
```

The next program, based on the one just listed, is somewhat more interesting. Whereas the chance of a single die falling with any face from one to six showing, is in theory one in six, the probability of certain totals being generated by adding the pips on two dice varies with the total being considered. That is, while there is only one way to get a total of 2 (1 plus 1), you can get a total of 4 by rolling 2 plus 2, 1 plus 3, or 3 plus 1. There are six ways to roll a 7. So, this next program simulates the rolling of two dice. Once you've seen it in action a few times, and you understand what is happening, add a routine to give you each total as a percentage of the number of throws. If all is well, your percentages should approach the theoretical distribution:

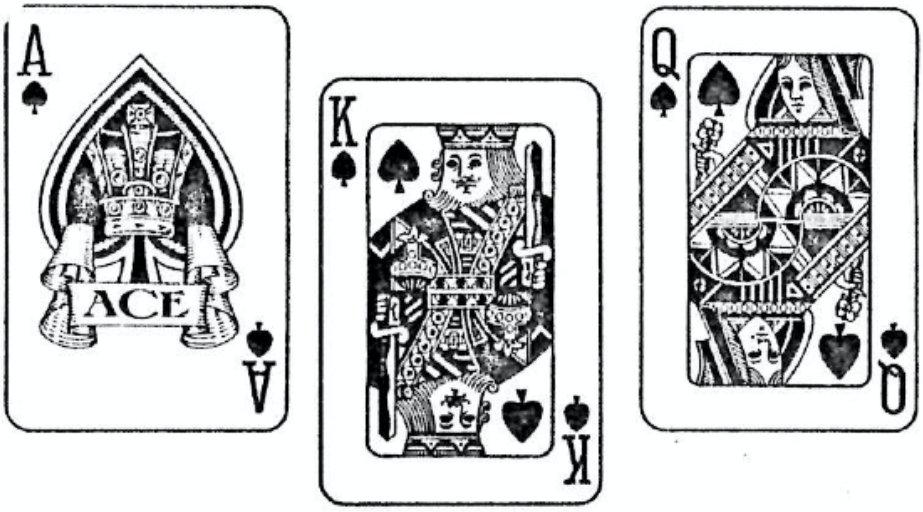
TOTAL SHOWING	NO. OF WAYS IT CAN BE THROWN	PROBABILITY	PERCENTAGE
2	1	1/36	2.77%
3	2	2/36, 1/18	5.55%
4	3	3/36, 1/12	8.33%
5	4	4/36, 1/9	11.11%
6	5	5/36	13.88%
7	6	6/36, 1/6	16.66%
8	5	5/36	13.88%
9	4	4/36, 1/9	11.11%
10	3	3/36, 1/12	8.33%
11	2	2/36, 1/18	5.55%
12	1	1/36	2.77%

```
10      IN.D;DIMAA12;F.J=1TO12;AA(J)=0;N.
20      F.C=1TOD
25      P.$12;?#E1=0
30      A=A.R.%6+1;B+A.R.%6+1;E+A+B
35      P."ROLL"C'"IS"E'
40      AA(E)=AA(E)+1
50      F.B=2TO12
60      P.B  AA(B)'
65      WAIT
70      N.;F.F=1TO600;N.;N.
80      END
```

Now that we've spent a bit of time emulating dice with the random number generator, I think we should use this in a few programs.

CHUCK-A-LUCK

The first one we'll look at is called CHUCK-A-LUCK. In its first incarnation, in England, it had the most improbable name of Sweat-Cloth, and when exported to the United States in the early years of the 19th century, it became known first as Sweat. Its name changed through the years to Chucker-Luck, Chuck-a-Luck or just plain Chuck. Nowadays, because of the equipment used in the non-Atom version, the game is often called THE BIRD CAGE.



The bird cage is an enclosed wire cage holding three dice. Players bet on the likelihood of a particular number coming up. If, for example, they place their money on six, and one of the three dice ends up with a six showing, they get their money back. If all three dice show six, they get back their money three times over. It is a fairly simple game, but one which arouses passion among bird cage devotees.

This program does most of the work for you. When it first asks SIZE OF BET? you enter the number of dollars you wish to bet. At all times you know how much money

you have left (you start with $30) and you can bet up to the total you have in hand. You automatically lose this amount as the game begins (so one die coming up with your number just returns your money, you need two or three to show a profit). Your next prompt will be NUMBER? and here the computer just wants you to enter a number between one and six. The game explains itself as the program progresses.

```
5       P.$12
10      M=30
```

M is your starting stake, and M your running cash total.

```
20      GOS. 280
```

Subroutine 280 prints out your money on the screen and incorporates a delay loop (290) which is accessed separately at one point (line 220).

```
30      IN."SIZE OF BET"A;IF A > M G.30
70      M=M-A
80      IN."NUMBER"B;IF(B < 1)/ (B > 6) G.80
```

The next section rolls the die, changing your stake as it does so if you win.

```
120     F.C=1TO3;W=0
```

W is the win/loss flag, and paymaster.

```
140     GOS. 290;D=A.R.%6+1
160     P."DIE"C'"FELL"D'
170     IF D=B  W=A;P."WIN $"W;M=M+W
```

The subroutine prints the total again.

```
210     GOS.280
215     N.
```

There is a short delay before the game continues.

```
220     GOS. 290
230     P.£12
```

If you have money, action moves back to line 20.

```
240     IF M> 0  G.20
```

If you are broke, woe...

```
250     P."GAME OVER, YOU ARE BROKE  ";G.250
```

And finally, this is the tally-changer, and delay, subroutine.

```
280     P."STAKE NOW IS £"M'
290     F.N=1TO60;WAIT;N.;R.
```

SEVENTH HEAVEN

In its youth as a dice game, this was known as UNDER AND OVER SEVEN. It seems very attractive to gamblers, because it appears loaded in the player's favour. As you'll discover when you play it with money that only exists in the Atom's variables store, the game is loaded against the player. If the random number generator worked perfectly, and you played this game for ever, your losses would outweight your gains by 16 2/3%. Now that you've been warned, we'll move into SEVENTH HEAVEN.

Once again we'll use M to represent your money.

```
20      M=30
```

And once again we have a subroutine to print out your growing (?) total.

```
30      GOS. 320
```

The next section asks you to place your bet on one of the three possibilities; that the dice will land with a total under seven (A); equal to seven (B); or over seven (C). You indicate your choice by entering either A, B or C.

```
40      P. "PLACE YOUR BET"'
50      PRINT "(A) UNDER 7  (B)7   (C) OVER 7"'
60      PRINT "  EVEN      5 FOR 1    EVEN"'
70      INPUT $R
```

Line 60 explains your winnings - money back (A or C) or 4 to 1 (B, written as 5 for 1 to look as if you are getting more...sneaky). If you wish to stop the game at this point, you enter S.

```
80      IF $R="S" END
```

Now you enter the amount of your bet.

```
90      IN."AMOUNT"Z
130     B=A.R.%6+1
150     C=A.R.%6+1
160     P. "THE DICE FELL "B" AND "C'
170     D=C+B;P."THE TOTAL IS "D'
```

This section of the program works out how well, or otherwise, you have done.

```
180     W=-Z
190     IF(D=7)&($R="B")W=4*Z
200     IF(D < 7)&($R="A")W=Z
210     IF(D > 7)&($R="B")W=Z
230     M=M+W
240     IF W > 0 P."YOU WIN $"W'
250     IF W < 0 P."YOU LOSE $" -W'
260     GOS. 320
270     F.N=1TO200;WAIT;N.
290     IF M < 1 END
300     P.$12;G.30
320     P."STAKE $"M';R.
```

CRAPS

In 'THE COMPLETE BOOK OF DICE GAMES', Skip Frey describes Craps as "the premier dice game". According to Mr Frey, "It is played everywhere, from back alleys to posh casinos in Las Vegas and Monte Carlo". Despite this glowing description, it can become a very dull game indeed when played with a computer. After all, to play the game you just roll dice, and if you have your trusty Atom to do this for you, there isn't much else to do.

Therefore, we've jazzed up the program a bit, to save you falling asleep at the keyboard. This Craps program gives you a starting stake of $20 and then adds to it, or takes away, in accordance with your luck with the dice.

OFFICIAL TERMS FOR CRAPS:

NATURAL - A 7 or an 11 on the first roll is a NATURAL. Roll this, and you win.

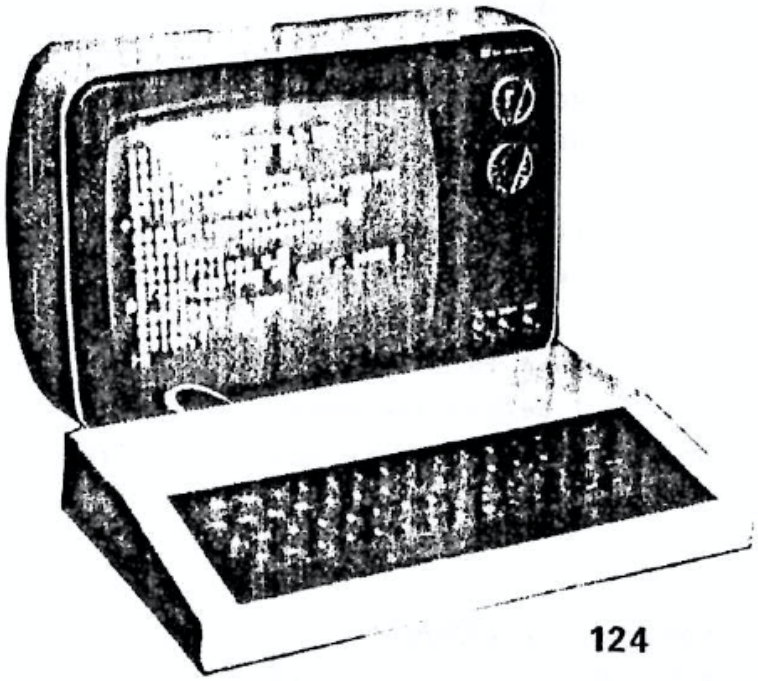
CRAPS - A 2, 3 or 12 on your first roll is CRAPS. Roll this, and you've lost.

POINT - A 4, 5, 6, 8, 9 or 10 on the first roll becomes your point. In the program, the variable E is your point.

If you don't roll a natural or craps, you continue tp roll until you 'make your point'. In this game, you make at least $6 if you make your point (see line 270). However, and this is a big however, you lose if you throw a 7 before you make your point (line 130). So long as you manage to make your point, the Atom will give you a new game.

```
5       DIM S2;A=0;P,S12;M=20
10      A=0;E=0;P."       STAND BY, PUNTER"'
```

```
50      GOS.320
60      B=A.R.%6+1;C+A.R.%6+1;D=B+C;A+A+1
70      IFA=6 P.$12;A=2;P."YOUR POINT IS "E'
100     IF A=1 G.150
110     IF D=E G.270
120     IF D=7 G.300
130     P."THE DICE CAME UP "B," C'"TOTAL  "D';
            G.50
150     IF(D=7)/(D=11)G.190
160     IF (D<4) / (D=12) G.210
170     E=D;P."YOUR POINT IS "E';G.130
190     P."YOU ROLLED "D" SO YOU WIN";M=M+5+A;G.220
210     P."FATE GAVE YOU A "D" SO YOU LOSE"';M=
        M - 3*A
220     P.';IF M>0 P."YOU'VE GOT $"M';G.225
222     P."YOU ARE BROKE"';M=20
225     P.';IN."ANOTHER GAME"$S;P.$12
250     IF $S = "YES" G.260
270     P."YOU ROLLED "D" AND MADE YOUR"'
        "POINT"';M=M+5+A;G.220
300     P."FOOL, YOU BLEW IT BY ROLLING "D';
        M=M-3*A;G.220
320     F.F=1TO6;P.$7;N.;?#E1=0;R.
```



GETTING TO GRIPS WITH ASSEMBLER

This section of the book is not intended to be a course in Assembler programming (the Atom manual covers that excellently) but as a guide to what Assembler is, and how and where you should use it.

Many people, even those who are conversant with BASIC, shy away from Assembler because it looks complex. But you'll find that once you get down to learn it, things quickly fall into place. Being able to program in Assembler is a rewarding experience and it can give you a degree of flexibility that you cannot attain in BASIC.

The Central Processing Unit (CPU) is the 'brains' of any computer. The Atom uses the well-tested 6502 microprocessor -- as do the PET and Apple computers. This CPU is programmed to carry out the instructions you give it (the program) but it does *not* understand BASIC. So this is where the Read Only Memory (ROM) comes in. The ROM contains the BASIC interpreter, which 'translates' your BASIC program into the language the CPU can understand -- machine-code. This process is called assembling and it is carried out by an Assembler program.

If you program in BASIC only, you do not need to know about the CPU and ROM, nor how they work. But if you intend to start programming in Assembler, an understanding of some of the fundamental processes is essential.

The Atom -- like all digital computers -- works in a binary number system. That is, a two-base system, where all numbers are expressed as zeros and ones. Human beings generally work in a decimal (or denary) system which is a base-ten system (using the numbers 0,1,2,3,4,5,6,7,8 and 9).

In the decimal system, we have individual single-digit

numbers up to nine. These would go into the units column (in the diagram below). If we take the number 10, the digit 1 goes in the 10s column and 0 goes in the units column. The number 10 really means (1*10)+(0*1) = 10. If we do the same thing for the number 245

100	10	UNITS	
		3	*(the number 3)*
	1	0	*(the number 10)*
2	4	5	*(the number 245)*

we see it is (2*100) + (4*10) + (5*1). The binary system works in exactly the same way, except that the columns are headed with different values. If we look at the same example in binary, we get:

128	64	32	16	8	4	2	1	
						1	1	*(the number 3)*
				1	0	1	0	*(10)*
1	1	1	1	0	1	0	1	*(245)*

The first example (3) is represented as 11:

(1*2)+(1*1) = 3

And our second example as 1010:

(1*8)+(0*4)+(1*2)+(0*1) = 10

It's not too difficult to recognise the lower numbers in their binary form, but as the numbers get higher, the binary number gets longer, as our next example shows:

(1*128)+(1*64)+(1*32)+(1*16)+(0*8)+(1*4)+(0*2)+(1*1)= 245

If we had a binary number of 11111111 it would represent the decinal number 255, the highest number that can be represented in binary with eight digits. You may have heard the term "eight-bit micro". This means that each character, number or symbol is stored as eight binary digits (bits) from 00000000 to 11111111.

The Atom would prefer its input in binary, and it actually coverts, more or less, whatever you input into binary before it works with it. This conversion takes a little time, and the conversion system takes up precious space in the ROM. It would be much simpler, and faster, if we could talk to the Atom in binary. But, as we have shown in the preceding examples, binary is a hassle for mere humans to manipulate. because of this, an in-between language, generally called HEX, has been developed.

HEX is short for hexadecimal, a base-16 system of numbering. We now need 16 different symbols to represent numbers. The numbers 0 to 9 are used, as well as the letters A to F to represent the numbers 10 to 15.

HEX	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DECIMAL	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

On the Atom, the HEX numbers are prefixed with a hash (#) symbol to tell them apart from decimal numbers. There is no real need to learn how to convert decimal numbers to their HEX equivalents (and vice-versa) because the Atom will do it for you:

```
PRINT &245
```

If you enter this on your Atom, then press return, the Atom will print #F4. The ampersand (&) tells the Atom that you want the HEX representation of 245 to be printed out. To convert numbers back from HEX to decimal, use a hash instead of an ampersand:

PRINT #F4 will give you 245.

It was mentioned earlier that 11111111 was the largest eight-bit binary number. This also corresponds to the size of each memory location on the Atom. This eight-bit size is often known as a 'byte'. A byte can hold a number from 0 to 256 (decimal), 00000000 to 11111111 (binary) or #00 to #FF (HEX). The fact that any eight-bit number

can be represented in HEX by a combination of any two symbols is why HEX is so popular and flexible to use.

The numbers held in a byte can be interpreted in three different ways:

i/ Simply as a number
ii/ As a representation of any one of 256 different characters
iii/ Or as an instruction for the CPU

When you program in assembler, the first byte of any instruction is what is called the operation code (or op code). Some instructions consist of just one byte - the op code. For example, #18 is the op code for *'clear carry flag'* and this instruction is self-contained; it does not need anything else. Some instructions specify addresses or other data following the op code. For example, #A580 means *'load the accumulator (#A5) into the address #80'*.

The Atom manual has a complete set of 6502 op codes in a separate section at the back.

The Atom computer is almost unique in that it allows you to mix both BASIC and Assembler in one program. If you want to start using Assembler in a program, the square brackets tell the Atom you are doing this. Even then, the Atom helps you further. You don't have to remember all the assembler instructions as a string of numbers, as there is a set of 'mnemonics' (aids to memory) to use. It is easier, and clearer, to use LDA #80 rather than #A580 for *'load the contents of the accumulator into address #80'*. When you write your Assembler program, the Atom will automatically convert the mnemonics into op codes -- and this process is called 'assembling' (which sounds either obscene or painful, depending on your disposition).

When as Assmbler program is RUN, the Assembler listing is printed on the screen. The mnemonics you've entered appear on the right of the screen, and the instruction op codes (and any instruction data or address) appear on the left.

The Assmbler printing out is very useful if you want to see what is happening, but it gets a bit tedious if it is printed out every time you play a game. It can be overcome by the use of a couple of control codes. Just before opening the square brackets to write you Assembler, enter another program line:

```
P.£21
```

This ends the output stream to the VDU and stops the Assembler listing from being printed. If the line:

```
P.£6
```

is entered just after closing the square brackets after the Assembler listing, the output to the TV screen will be turned on again.

Once the program has been assembled, you want to know where the Atom has stored it, in order to use it. If you add the line:

```
DIM P(-1)
```

before the Assembler program, the Atom will put the assembled machine-code at memory locations starting at TOP. Now you know where the machine-code is stored, you can access it with the command:

```
LINK TOP  (often abbreviated to LI. T.)
```

You should, by now, have a general idea of what Assembler is, and how it works. This is just about the right time to turn to the manual, and get to work on learning how to program with Assembler mnemonics. We've left program examples out of this section of the book, because everything is explained in the manual. However, the next section on keyboard scans has examples of a couple of Assembler routines. This illustrates very well the flexibility of using Assembler rather than BASIC. Atom BASIC does not have a GET£ or INKEY£ command as many other BASICs do, but this problem can be overcome by using Assembler. This is only an example of the many uses of Assembler.

As well as being more flexible, Assembler programming is much faster than BASIC. When you program in Assembler, the Atom does not have to spend time converting a program into a language it understands. It is also possible to access and use various

routines in the monitor once you know their addresses. An Assembler program is usually (but not always) more compact than its equivalent in BASIC and therefore occupies less space, an important consideration when memory is tight.

KEYBOARD SCANS

As we said a few moments ago in the section on Assembler, there is one useful BASIC command that does not appear in the Atom BASIC interpreter -- and that is a GET$ or an INKEY$ command. It is a handy function to have, replacing the need for an input which has to be followed with a RETURN before the computer will recognise it.

Whereas INPUT will wait until a value has been entered and RETURN pressed, an INKEY$/GET$ function will scan the keyboard until a key is pressed and then continue executing the program. A good example of the need for a keyboard scan is the game of SPACE INVADERS. There is no ATOM BASIC command to allow you to continually move an object on the screen under keyboard control while the program is running. So how, in the case of SPACE INVADERS, could you move your laser baser and shoot down the aliens on an Atom?

There are several answers, so we will look at a few of them here. Since you probably know more BASIC than you do Assembler (despite the preceding section), we will look at the BASIC keyboard scan first.

Section 25.4 of the Atom manual deals with the Input/output port allocations of the 8255 Programmable Peripheral Interface Adaptor. It tells you that the PPIA has three eight-but ports:

Port A - #B000
Port B - #B001
Port C - #B002

It does not matter whether you understand what ports are or not, they can be used as a limited keyboard scan. Each of the three addresses contains a value (or values) when no key on the keyboard is depressed. As soon as certain keys are touched (except, confusingly, in the case of #B000), the value will change as long as the key is held down. As soon as you take your finger off, the value returns to what it was originally. The original values in the three addresses are:

#B000 - 0
#B001 - 255
#B002 - 199/215/207

Try this next line to print out the values for all three addresses for yourself:

```
DO P.?#B001;U.0
```

If you run this line using *#B001* as the address, you should get lots of 255s printed on the screen. Press the SHIFT key down and the screen gets taken over by 127s. Take your finger off SHIFT and back come the 255s. There are only a few keys that give different values with #B001, and these are:

SHIFT - 127
CTRL - 191
3 - 253
= - 251
G - 247
Q - 239

If you press two or more keys down at the same time, you can get different values. Address #B000 does not have any keys that affect it, and #B002 is limited to one key (REPT which changes 215/207/199 to 135/151 when pressed).

Try entering this short program to see how you can use these addresses as keyboard scans:

```
10    IF?#B001 = 127 P."ACORN"
20    IF?#B001 = 191 P."ATOM"
30    P."(FIVE SPACES)";G.10
```

When you run this program, you'll see the Atom will print a series of blocks of five spaces, because the conditions specified in lines 10 and 20 are false. Without any keys being pressed, the value of ?#B001 is 255. As soon as you press the SHIFT key the word ACORN is printed on the screen, and will be printed over and over again so long as you hold the key down. Then as soon as you take your finger off the key the Atom returns to printing spaces. Try pressing CTRL. Now the word ATOM appears instead. What happens if you press both together? Neither ACORN now ATOM is printed because the value in address #B001 is neither 127 nor 191 when both keys are held down.

We can take this one step further and actually control the movement of an object on the screen. Try this:

```
10    A=16
20    IF?#B001 = 127 A = A + 1
30    IF?#B001 = 191 A = A - 1
40    A?#8010  = 127
50    P.'
60    G.20
```

The Atom will POKE an inverse space into the 16th square of the top line of the VDU (line 40). Line 50 scrolls the screen up, and then the inverse space gets POKEd into the top line again. This gives the flickering effect.

The position of the flickering inverse space is due to A having the value 16. If you press the SHIFT key down, line 20 adds one to the value of A and the inverse space gets POKEd one square to the right. Pressing CTRL moves the inverse square back to the left. If you keep the SHIFT key down the inverse space will move off the right of the screen and reappear on the left hand side, a line lower down.

If you keep the inverse space moving down the screen, you will get some interesting wiggly effects. Try it and see. This POKE and SCROLL idea with SHIFT and CTRL to move an object across the screen forms the basis of the racing car game GRAND PRIX SPECIAL elsewhere in this book. However, there is another way, and a better way, of producing a similar result by POKEing.

```
10    A=16
20    IF?#B001 = 127 A=A+1;A?#8000=127;B=A-1;B?#8000=32
30    IF?#B001=191 A=A-1;A?#8000=127;B=A+1;B?#8000=32
40    G.20
```

As you'll see when you run it, this gets rid of the flickering and the 'images' that the POKE and SCROLL routine produced. Notice how quickly the computer responds to you pressing the keys. You will probably have to add delay routines to slow the thing down. Adding another line to the program into the basis of a SPACE INVADERS program (even if it is upside down).

```
35    IF?#B001=247;C=A+32;FOR F=1TO12;C?#8000=46;
      C=C+32;N.
```

Press G to fire (or drop your bombs). Now that you've seen this simple program in action, demonstrating the BASIC keyboard scan, you could try and develop a game of SPACE INVADERS (or a simplified version of it). Combine the above program with what you've learned from the MASTERING THE GRAPHICS and the PEEK and POKE chapters and create a programming masterpiece.

We can use the ideas we've discussed to produce a pretty simple MOONLANDER. This is how we went about it. First enter these lines:

```
1     P.#12;@=0;CLEAR 0;Q=ABSRND%9*10+200;P.Q'
10    FOR F=0TO31;F?#81E0 =127;N.
20    FOR F=1TO60;WAIT;N.
```

```
25    X=ABSRND%25;X?#81C8=#FF
30    A=ABSRND%20+7;A?#8000=30
```

These first five lines of the program set up the variables for the start of the game. Q is the amount of time you have to land the spaceship, and you can vary this if you think you need more (or less) time. X is the position of the mother ship on the surface of the moon and A is the position on the top line where your spaceship first appears.

```
100   IF?#B001=127 B=1+1
110   IF?#B001=191 B= -1
120   IF?#B001=247 B=- 64
```

This is where the keyboard scan comes into the program. Three keys are being used - SHIFT (127), CTRL (191) and G (247). SHIFT gives your scoutship lateral thrust (!) to the right, CTRL gives it left lateral thrust and G fires the main boosters.

```
130   Q=Q-10;IFQ =0 G,300
140   P,$11;?#E1=0;P,Q"    "'
```

These two lines decrement your score and print it in the top left hand corner of the screen. P.$11 is a very interesting command. It moves the cursor back up one line on the screen. *?#E1=0* turns the cursor off, so P.Q" "' has the effect of over-printing the previous line. The extra spaces are the Q are to erase the last zero from a three three-digit number when the time falls below 100. Look at the time changing as the program is running.

```
140   A=A+B;FOR F=1TO20;WAIT;N.;A?#8000=32;A=A+32;
      A?#8000=30
142   E=A-B-32;E?#8000=32
```

Lines 140 and 142 move your scoutship around. A blank square is POKEd into the square where your scoutship was, and your scoutship is POKEd into another square, depending upon which you direction you moved.

```
145   D=A+32;IF D?#8000=127 G.300
146   IF D?#8000=42 G.400
150   G.100
```

The above lines check whether the scoutship has crashed onto the surface of the moon, in which case it goes to the routine at 300; whether it has docked safely with the mother;ship, in which case it goes to the routine at 400; or -- if neither of the above have happened -- goes back to the keyboard scan.

```
300   P.&7;A?#8000=158;FOR F = 1 TO 20;WAIT;N.
310   A?#8000=#66;FOR F= 1 TO 20;WAIT;N.
320   A?#8000=#41;FOR F=1TO20;WAIT;N.
330   A?#8000=32;END
```

The above four-line routine provides a bleep (P.&7) and a pretty graphical display of your scoutship breaking up. This routine is called when your scoutship hits the surface of the moon, or you run out of time.

```
400   FOR F=1TO10;P.';N.
410   P. "SAFE DOCKING" ';P.&11; ?#E1=0
420   FOR F=1TO20;WAIT;N.
430   P."SAFE";P.&128;P."DOCKING";P.&11;?#E1=0
440   FOR F=1TO20;WAIT;N.;G.410
```

This last routine is called if you manage to dock your ship successfully with the mother-ship. You get the SAFE DOCKING message flashed at you using the same principle as the time...moving the cursor up a line and over-printing. This produces a very good effect.

This program is really only intended to demonstrate the use of the BASIC keyboard scan, where all you want to do is move an object across a screen, and perhaps fire lasers. If you want a more sophisticated keyboard scan, where practically every key is scanned and returns its unique value, then you must turn to Assembler.

You don't have to understand Assembler to use this keyboard scan, although it is interesting to follow what is going on. The magic number to remember is *#FFE3*, because we are sure that once you know what you can do with it, there will be few programs you write where you cannot find a use for it.

The Atom's monitor already contains routines for reading characters from the keyboard, and for printing characters on the screen. These routines can be accessed via short Assembler programs.

Two of the more useful routines are OSWRCH (*#FFF4*) and OSRDCH (*#FFE3*) which are acronyms for Operating System WRite CHaracter, and Operating System ReaD CHaracter. The Atom manual shows you how to use the OSWRCH routine, but makes no mention of OSRDCH. It is this routine, starting at address *#FFE3*, that is the pivot of the scan. The OSRDCH routine, when called, waits for a key to be pressed, and then immediately stores the code of that key in the accumulator. You can still use this routine without worrying about the value stored in the accumulator. Try entering the following program:

```
10    P."PRESS ANY KEY"'   '
20    LINK #FFE3
30    P."A KEY HAS JUST BEEN PRESSED"'   '
40    GOTO 10
```

When you run the program, as it obvious, the computer asks you to press any key. As soon as the key is pressed a message telling you this is shown on the screen.

LINK #FFE3 will cause the Atom to stop the execution of the program until any key on the keyboard (except CTRL, SHIFT, LOCK, REPT and the cursor control keys) is pressed. This is a very useful routine when you want to stop the execution of a program without requiring an input. It is used in the POETRY program in the book, among others.

But what if you want a value to be enter and subsequently manipulated? Here you'll have to resort to a full Assembler program.

```
10    DIM P(-1)
20     [
30    JSR #FFE3
40    STA #80
50    RTS
60     ]
70    LINK TOP
80    PRINT ?#80
90    GOTO 70
```

When you run this program, you should get the Assembler listing printing on the screen.

```
20   2949
30   2949 20 E3 FF      JSR #FFE3
40   294C 85 80         STA #80
50   294E 60            RTS
```

The first column of numbers is the program line numbers. To the right of these are the addresses where the assembled machine code is store (immediately after the program text in this case). The numbers (in HEX) are for the expanded Atom. If you have an unexpanded machine, these values will be around #8200.

The next column of numbers (HEX) is the intsruction op codes. For example; *#20* means JSR. The other two columns of numbers are the instruction data/addresses.

The mnemonics you typed in appear on the right of the screen. Let us look at how the Assembler program works;

i) *DIM P(-1) is a BASIC command which tells the Atom to store the assembled machine code in memory locations starting immediately after the program text*

ii) *[- opening square brackets tells the Atom that you are now going to use the assembler language*

iii) *JSR #FFE3 - jump to the subroutine at address #FFE3. This is the OSRDCH routine that stores the value of the key pressed in the accumulator*

iv) *STA #80 - store the value in the accumulator in address #80. Address #80 is not used by the Atom. It is one of a group of addresses (#80 to #AF) that have been set aside as workspace.*

v) *RTS - RETURN to BASIC when the Assembler program has finished.*

vi) *] - closing square brackets tells the Atom that you have finished using Assembler and are now going back to BASIC.*

The program can be written much more neatly in one line, by using the fact that the Atom can handle multi-statement lines:

```
20      [;JSR #FFE3;STA #80;RTS;]
```

And you can now get rid of lines 30 to 60. The Assembler listing now looks like this:

```
20    2941
20    2941 20E3 FF     JSR #FFE3
20    2944    65 80    STA #80
20    2946    60       RTS
```

As you can see from the memory locations, this second way only takes up eight fewer bytes.

Now that the machine code has been assembled, you have to be able to get access to it to use it. This is what the command LINK TOP (or LI.T.) does. LINK TOP calls the routine stored starting from TOP (which is the address immediately following the program text) where we have stored our keyboard scan routine. In this case, it is very much like using *LINK #FFE3*. The computer waits until a key is pressed, but this time the value of the pressed key that is placed in the accumulator is transferred to address *#80 (STA #80)*. Line 80 PEEKs the value stored in

address *#80* and prints it.

This routine offers great scope for use in programs, especially in the situation where you have to make a choice. For example, add these lines:

```
15    P.&21
40    P.&6
50    A=ABSRND%3+1
60    P."GUESS MY NUMBER 1 - 3"';P.'
70    LINK TOP
80    IF A=?#80-48 P."CORRECT "';P.'G.50
90    P."SORRY, IT WAS "A';P.';G.50
```

Line 50 chooses a random number between one and three. There is no need for an input as LINK TOP activates the keyboard scan, and the computer will wait until a key is pressed. As soon as a key is pressed it checks the number it has chosen (A) with the number you guesses (stored in address *#80* as a code). That is why 48 is taken away from the stored value. The code for one is 49, for two it is 50 and for three it is 51. A full list of codes can be found at the back of this book.

There is one drawback of using this Assembler keyboard scan. Calling subroutine *#FFE3* tells the Atom to wait until a key is pressed before continuing with the program. This may be OK for a program such as the number guess listed above, but it is certainly not suitable for games and programs with a 'real time' element.

It is no good playing SPACE INVADERS if you have plenty of time to work out what you are going to do.

So we have to turn to another type of keyboard scan. This second one is also an Assembler program. It looks very similar to the previous program, except that we call the routine at *#FE71* (the routine the Atom actually uses to check which key is pressed when INPUT is reached) instead of *#FFE3*. This routine stores the code of the

key pressed in the register Y instead of the accumulator, so *STA #80* has to be changed to *STY #80*.

However, the codes stored this time are of different value to those from the previous routine. You can run the following program to tell you which codes go with which keys.

```
10    DIM P(-1);@=0
20    [ ;JSR #FE71;STY #80;RTS; ]
30    LINK TOP
40    P."CODE FOR KEY PRESSED IS "?#80
50    G.30
```

Keep your finger pressed down on the "A" key when you run this program. You'll get the message CODE FOR KEY PRESSED IS 33 printed over and over again. This happens because the routine *#FE71* does not wait until a key is pressed, unlike routine *#FFE3*. This short program should explain things clearly:

```
10    DIM P(-1)
20     [;JSR #FE71;STY #80;RTS;]
30    LINK TOP
40    P."ACORN ";IF?#80=33 P."ATOM  "
50    G.30
```

As the program runs, line 30 calls the routine at *#FE71*, and this routine determines whether a key is pressed or not, and stores the value in the Y register in address *#80*.

Since routine *#FE71* does not require the Atom to wait for a key to be pressed, the rest of the program is executed. Line 40 prints the word ACORN, and if the value of 33 has been stored in address *#80* (that is, if the A key has been pressed) the word ATOM is printed as well. In this case it hasn't, so line 50 sends the computer back to line 30 and the process is repeated, Try pressing the A key

key down and see what happens. ACORN is always printed, and ATOM is only printed if the A is held down.

The following program demonstrates the use of this keyboard scan. It is a very simple program. Your car (a greater than symbol) is in the top left hand corner of the screen, and you are in a hurry to get home (to the bottom right hand corner). Can you make it home without crashing into other cars. Use the keys Q (diagonally upwards), A (straight ahead) and Z (diagonally downwards).

This program uses the PEEK and POKE functions (explained in detail in the PEEK and POKE section of the book) to change the screen display, and the second Assembler keyboard scan program to produce a continuous action game.

```
10    DIM P(-1); P.$12;P.$21
20    [;JSR #FE71;STY #80;RTS; ];P.$6
```

These first two lines of the program clear the screen (P.$12) and assemble the machine code keyboard scan. DIM P(-1) tells the Atom to store the machine code routine at TOP (the first free address after the program listing). P.$21 turn the output to the VDU off and on respectively to stop the Assembler listing from appearing on the screen.

```
50    FOR F=1TO60;WAIT;N.
60    FOR F= 0 TO 318;K=A.R.?20;IF F?#8000=190
      F?#8000 = 190
70    IF K = 5 F?#8000=159
80    N.
```

After a brief pause in line 50, the above routine starts building up the traffic. There's a check in line 60 so make sure your car is never obliterated by an Atom car.

```
90    LINK TOP;GOS.a
```

Line 90 calls the keyboard scan. Unlike the *#FFE3* scan,

this routine does not wait until a key is pressed but only checks to see if a key is pressed while the routine is being called. We will jump over the next few lines of the program and look at subroutine a.

```
500a  FOR F=1TO10;WAIT;N.
510   IF?#80=53 Y=1;R.
520   IF?#80=49 Y=-31;R.
530   IF?#80=58 Y=33;R.
540   R.
```

The short wait routine is included to slow the game down just a fraction. Without it the game runs so fast you cannot follow it. Lines 510 - 530 check to see if keys A, Q or Z are pressed, and if one of them is being held down, a value is assigned to Y and the execution of the program returns to line 100. If none of the keys are being pressed, line 540 sends execution back to line 100 without changing the value of Y (that is, your car continues to move in the same direction).

```
100   ?X=127;X=X+Y;IF?X=159 G.b
110   IF?X=15 G.c
120   ?X=190; G. 60
```

Line 100 POKEs your car's old position with a white square and then checks to see if you have hit one of the Atom's cars (and sends you to subroutine b if you have). If you have not crashed or arrived home, line 120 POKEs your car into its new position and goes back to line 60 to call the keyboard scan again.

```
550b  FORF=1TO100;WAIT;?X=#66;WAIT;?X=#59;N.
      ?X=32
560   P."CRASH"''&11"CRASH''&11;G.560
```

Subroutine b frantically POKEs alternating graphical symbols into the square where your car crashed, and then prints the word CRASH until the end of the world.

```
570c  ?x=190;p,#7
580   p."SAFE ARRIVAL"'#11"SAFE ARRIVAL"'#11;g.580
```

This last subroutine puts your car in the garage and flashes up the SAFE ARRIVAL message.

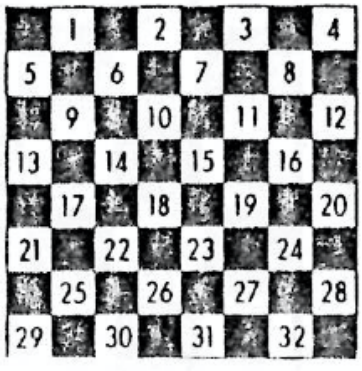
These keyboard scan routines are used in other programs in the book, and by examining the listings, you'll see how they can be used to provide useful alternatives to the conventional INPUT- RETURN.

DRAUGHTS - AND OTHER BOARD GAMES

This next section will enable you to develop a complete game of draughts. However, the main reason it has been included is so we can explain a method of numbering boards for board games which are easy for a computer to manipulate.

A similar board-numbering system can be used as the core of a chess program, noughts and crosses and even Nine Mens Morris. We suggest you read through the text carefully, and not just enter the final game, because to do so would rob you of much of the value of this section.

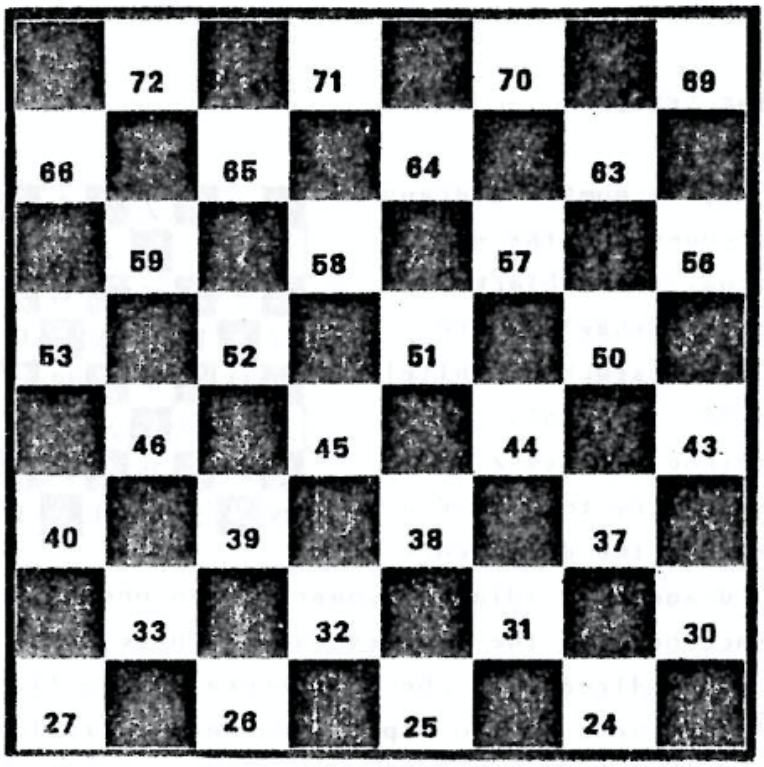
THE NUMBERING SYSTEM



The normal way to number a draughts board is to count off the white squares (actually the black ones are counted, but they're hard to see, so we've stuck to white) from one to 32. But this numbering method creates a problem when we try to define a move in terms of the difference between two diagonally adjacent squares. In one direction, the difference between the squares can be three or four, and in the other direction, the difference can be five or four. Also, there are no 'spare' numbers to indicate where the edge of the board begins.

To get around these problems, a certain A. L. Samuels wrote an article for Scientific American in the 1960s (see Strachey, Christopher *Systems Analysis and Programming* in Readings from Scientific American, W. H. Freeman and Co., San Francisco, 1971) in which he devised a clever numbering system in which the difference between diagonally adjacent squares was always four and five (or minus four and five). It also allowed for numbers to be given to squares which were 'off the board'. We've changed his numbering system a little to make it more convenient for the Acorn Atom, and in our system, the difference between squares is always six or seven (or minus six or seven). The system, very simply, sets up an array, and allots certain elements of the array to squares on the board. All other elements of the array are understood by the computer to refer to squares which are off the board.

In this program, the Atom allots the value 9 to any square off the board, zero to an empty square, 1 to a computer's piece and -1 to a player's piece. This may sound a little complicated, but bear with us, and (hopefully) all will become clear.



As you'll see from our numbered board, if you move from the top left hand corner (69) to the square diagonally below it (63), the difference between the two squares is -6. Now, choose any other square on the board from which you can move down and to the left, and you'll see there is a difference of -6 between the square you started on and the square on which you'll finish. This sort of predictable result is relatively easy for a computer to handle.

Move in the other direction, that is downwards and to the right, and you'll see the difference between the two squares is -7. In the first version of draughts, we'll actually be playing on this printed board, so you'd better starting looking for some tiny buttons to use as playing pieces (or transfer the numbers onto a bigger board).

The program is in two parts. The first 'sets up the board' (the subroutine starting at 9000) and the second actually plays the game. Your pieces start at the bottom of the printed board (on the lower numbers) and the Atom's start at the top. The Atom's moves are shown as two numbers. The first is the number of the square it is moving from and the second -- natually enough -- is the square it is moving to. Move its piece on the board as instructed and then decide on your move. Make sure you move your piece before you enter your move (which you do by entering the number of the square you're moving from, then RETURN, then the number of the square you're moving to) or you may forget what your move was. The Atom will sometimes make a decision so quickly you barely have time to take your finger off RETURN before it has moved. Note that there is no provision within the program for multiple jumps or for the making of kings. This first draughts program is just to give you a game you can enter fairly easily, which will help you to understand how the Atom will play in the final version.

Here is the first part of the program, the section which

sets up the board. It creates an array (AA) then fills the array with numbers representing pieces on the board (1 and -1) empty squares (0) and squares off the board (9).

```
1     GOSUB 9000
9000  DIMAA82;DIMXX2;XX1=-6;XX2=-7
9010  ?#E1=0
9050  F.Z=1TO82;AA(Z)=9
9070  IF((Z<73)&(Z>55))&(Z<> 67)&(Z<> 68)&
      (Z<>69)&(Z <> 61)AA(Z)=1
9080  IF((Z< 54)&(Z>42))&(Z<>47)&(Z <> 48)&(Z<>49)
      AA(Z) = 0
9090  IF((Z< 41)&(Z>23))&(Z<>34)&(Z<>35)&(Z<> 36)
      &(Z<>28) AA(Z)= -1
9100  N.
9110  AA62=9;AA29=9;AA60=9;R.
```

Check to see if this is working correctly before you proceed. Add 8000 STOP and run the program. When it stops, get the Atom to print out the following, to make sure it has assigned values correctly. The right answer is given in brackets after the command:

```
PRINT AA23      (9)
PRINT AA54      (9)
PRINT AA64      (1)
PRINT AA38      (-1)
PRINT AA51      (0)
PRINT AA73      (9)
```

If this works correctly, proceed to enter the following which actually plays the game.

```
10w   Q=0;P.#12
15    ?#E1=0
20    Z=24
30    IF((AA(Z)=9)/(AA(Z)=0)/(AA(Z)= -1))G. m
40    ?#8000=RND%255
50    X= 1
60    IF(AA(Z+XX(X)) < 0)&(AA(Z+2*XX(X))=0) Q=XX(X)
```

```
80    IF Q<>0 G.120
90    IFX=1 X=2;G.60
105m  IF(Z<73)Z=Z+1;G.27
110   IF Q=0 G.160
120   AA(Z+Q)=0;AA(Z+2*Q)=AA(Z);AA(Z)=0
150   P."MY MOVE"Z, Z+2*Q
155   G.295
160   Y=0
170   Y=Y+1;?#8015=Y;Z=  A.R.%48 + 24
190   IF((Y<100)&(AA(Z)<> 1))G.170
200   X=1
210   IF(AA(Z+XX(X))=0) Q=XX(X)
230   IF Q<>0 G.290
240   IFX=1 X=2 G.210
260   IF Y< 100 G.170
270   P."YOU WIN";END
290   AA(Z+Q)=AA(Z);AA(Z)=0
292   P."MY MOVE"Z ,Z+Q'
295   INPUT"YOURS FROM"A
300   IFA=99 END
330   INPUT"TO"B
340   AA(B)=AA(A);AA(A)=0
360   IF (ABS(A-B)>7);T=(B-A)/2;AA(A+T)=0
370   G.W
```

Play a game through with the computer, and see if you can work out how it makes its moves. Having done this, you can add a routine (developed by Tony Baker) to print out the board. Delete line 295, change 292 and 150 to GOSUB 5000, and delete line 40. Now add the following:

```
9010  @=0;?#E1=0
5000  F.H=24TO72
5010  IFAA(H)=1 AA(H)=111
5020  IFAA(H)=-1 AA(H)=120
5040  IFAA(H)=0 AA(H)=128
5050  N.
5055  P.£30
5060  PRINT'MY MOVE  "Z" TO "2*Q';P.''
5090  F.K=0TO3;F.J=0TO3
5110  P."space"£(AA(72-J-13*K))
```

```
5120  N.J
5140  F.J=0TO3
5150  P.$(AA(66-J-13*K))"space"
5160  N.J;P.';N.K
5200  F.H=24TO72
5210  IFAA(H)=111 AA(H)=1
5220  IFAA(H)=120 AA(H)= -1
5230  IFAA(H)=128 AA(H)=0
5250  N.
5260  R.
```

Ignore the line "MY MOVE..." when it gives the same square number twice. Wait for it to change, and then watch the pieces move. Note that the two loops (5000 to 5050 and 5200 to 5250) decide which character will represent which piece. By all means change these if you like. (And for a bit of light relief, add 55 *?#8008 = RND;?#8009=RND*).

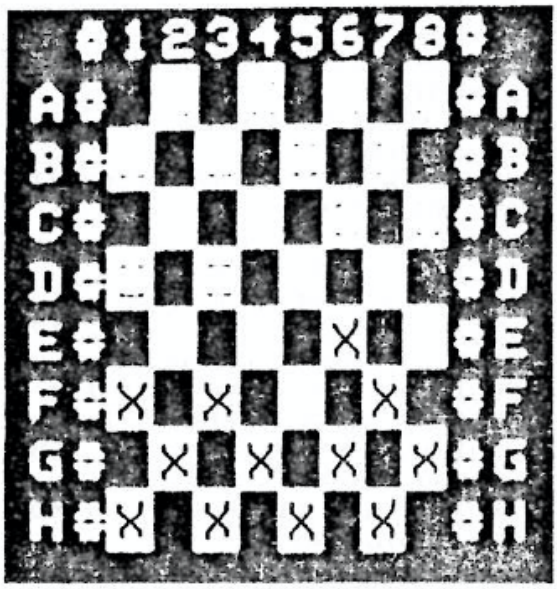
Now follows a complete draughts program (kings and multiple jumps). This game uses the same board as the the game you have just entered, and you should be able to follow the listing through using the material given in this section.

After the draughts listing, you'll find a variation of draughts which the Victorians used to play. If you enter the draughts game, you'll find you can easily get the variant by deleting certain lines of the draughts game. You should find the Atom plays the variant game better than it plays draughts.

We'll leave it to you to work out why.

DRAUGHTS

```
   5P. $12
  10GOS. 9000
  15P. $7    $7
  50M=0; GOS. 7000
1000P. '    "LAST MOVE "$G'
1010M=0
1012F. I=1TO2
1015IFI=1INPUT"FROM"$E
1030IFI=2INPUT"TO"$E; $G=$E
1040IF$E="A2"FF(I)=72
1110IF$E="A4"FF(I)=71
1120IF$E="A6"FF(I)=70
1130IF$E="A8"FF(I)=69
1140IF$E="B1"FF(I)=66
1150IF$E="B3"FF(I)=65
1155IF$E="B5"FF(I)=64
1160IF$E="B7"FF(I)=63
1170IF$E="C2"FF(I)=59
1180IF$E="C4"FF(I)=58
1190IF$E="C6"FF(I)=57
1200IF$E="C8"FF(I)=56
1210IF$E="D1"FF(I)=53
1220IF$E="D3"FF(I)=52
1230IF$E="D5"FF(I)=51
1240IF$E="D7"FF(I)=50
1250IF$E="E2"FF(I)=46
1260IF$E="E4"FF(I)=45
1270IF$E="E6"FF(I)=44
1280IF$E="E8"FF(I)=43
1290IF$E="F1"FF(I)=40
1300IF$E="F3"FF(I)=39
1310IF$E="F5"FF(I)=38
1320IF$E="F7"FF(I)=37
1330IF$E="G2"FF(I)=33
1340IF$E="G4"FF(I)=32
1350IF$E="G6"FF(I)=31
1360IF$E="G8"FF(I)=30
1370IF$E="H1"FF(I)=27
1380IF$E="H3"FF(I)=26
1390IF$E="H5"FF(I)=25
1400IF$E="H7"FF(I)=24
1405IF(I=1)&(AA(FF(1))=B)G. 1015
1406IF(I=2)&(AA(FF(2))<>B)G. 1030
1410N. I
1420M=0; AA(FF(2))=AA(FF(1)); AA(FF(1))=B
1460IFABS(FF(1)-FF(2))>7; M=1; AA(FF(1)+((FF(2)-FF(1))/2))=B
1470GOS. 7000
2002Q=0; Z=24; D=1
2010IF((AA(Z)=H)&(Z>68)&(Z<73))AA(Z)=W
2020XX1=-6; XX2=-7; XX3=6; XX4=7
2021IFAA(Z)=120 G. 2110
2022IFAA(Z)=119 G. 2110
2023IFAA(Z)=9 G. 2110
2024IFAA(Z)=128 G. 2110
2030IF((AA(Z)=C)&(Z>23)&(Z<28))AA(Z)=K
```



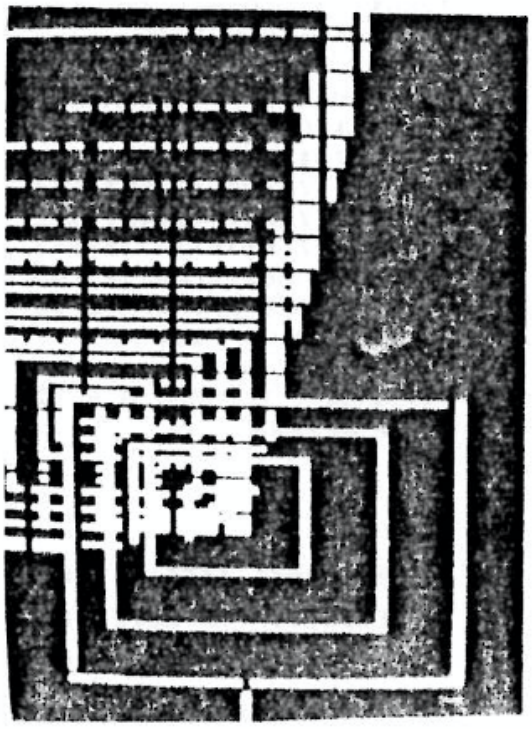
```
2045U=0;IF (AA(Z)<>K)&(D>2) G.2110
2050IF(AA(Z+XX(D))=H)U=7
2052IF(AA(Z+XX(D))=W)U=7
2055IF((U=7)&(AA(Z+2*XX(D))=B))Q=XX(D)
2056IFQ<>0G. 2130
2070IF((AA(Z)=111)&(D>2))G. 2110
2090IF(D<4)D=D+1;G. 2045
2110IF(Z<72)Z=Z+1;D=1;G. 2010
2120IFQ=0G. 2240
2130AA(Z+2*Q)=AA(Z);S=S+1;AA(Z+Q)=B;P=Z+2*Q;AA(Z)=B
2140GOS. 7000;F. O=1TO15;WAIT;N.
2160N=0;D=1;U=0
2165IFAA(P+XX(D))=B G. 2200
2170IF((AA(P+XX(D))=H)\(AA(P+XX(D))=W))U=7
2172IFU=0G. 2200
2174IF((AA(P)<>K)&(D>2))G. 2210
2175IF((U=7)&(AA(P+2*XX(D))=B))N=  XX(D)
2190IFN<>0G. 2220
2200IFD<4 D=D+1;G. 2165
2210IFN=0G. 50
2220AA(P+N)=AA(P);AA(P)=B;S=S+1;AA(P+N/2)=B;G. 50
2240Z=72;U=0
2250IF((AA(Z)<>C)&(AA(Z)<>K))G. 2290
2255?#8016=RND
2260IF(AA(Z-13)=H)G. 2290
2262IF(AA(Z-12)=H)G. 2290
2265IF(AA(Z-14)=H)G. 2290
2270IF  (AA(Z-6)=B)&((AA(Z-12)=C)\(AA(Z-12)=K))U=-6
2275IFU=-6&AA(Z-18)=H Q=Z-6
2280IF(AA(Z-7)=B)&((AA(Z-14)=C)\(AA(Z-14)=K))U=-7
2285IFU=-7&AA(Z-21)=H Q=Z-7
2290IF(Q=0)&(Z>23)Z=Z-1;G. 2250
2350Y=0
2360Z=24+A. R. %48;Y=Y+1;IFY>400G. 2440
2370?#8018=RND
2380IF((AA(Z)<>C)&(AA(Z)<>K))G. 2360
2382IF((AA(Z)=K)&(Z>55))G. 2360
2385IFY>200G. 2390
2386IF(AA(Z-13)=H)G. 2360
2387IF(AA(Z-14)=H)G. 2360
2388IF(AA(Z-12)=H)G. 2360
2390D=1
2395IF((AA(Z)=107)&(RND%5>-3))D=3
2397IF((AA(Z)=107)&(Z>55))D=1
2400IF((AA(Z)=C)&(D>2))G. 2360
2410IFAA(Z+XX(D))=B Q=XX(D);G. 2460
2420IFD<4 D=D+1;G. 2400
2430G. 2360
2440$E="Z";G. 7000
2460AA(Z+Q)=AA(Z);AA(Z)=B;G. 50
7000P. $30;@=0
7010?#E1=0
7020H=120;C=111;W=119;K=107;B=128
7030IFM=1;T=T+1
```

```
7100P. "       #12345678#"
7110P. "      A# "$AA72" "$AA71" "$AA70" "$AA69"#A"
7120 P. "      B#"$AA66" "$AA65" "$AA64" "$AA63" #B"
7130P. "      C# "$AA59" "$AA58" "$AA57" "$AA56"#C"
7140 P. "      D#"$AA53" "$AA52" "$AA51" "$AA50" #D"
7150P. "      E# "$AA46" "$AA45" "$AA44" "$AA43"#E"
7160 P. "      F#"$AA40" "$AA39" "$AA38" "$AA37" #F"
7170P. "      G# "$AA33" "$AA32" "$AA31" "$AA30"#G"
7180 P. "      H#"$AA27" "$AA26" "$AA25" "$AA24" #H"
7210IF$E="Z"T=12
7220IF(T=12)P. "YOU WIN";END
7230IF(S=12)P. "I WIN";END
7240IFM=0P. "                                    "
7260$L="";IF M=1 INPUT"CAN YOU JUMP AGAIN (Y OR N)"$U;M=0
7310IF$L="Y"G. 1000
7320R.
9000DIMAA86 ;Y=0;DIMXX4;XX1=-6;XX2=-7;XX3=6;XX4=7;DIML1;DIMFF2
9060H=120;C=111;W=119;K=107;B=128;Q=0      9070 DIM G2,E2
9110F. Z=1TO86;AA(Z)=9
9130IF((Z<73)&(Z>55))&(Z<>67)&(Z<>68)&(Z<>61)AA(Z)=C
9140IF((Z<54)&(Z>42))&(Z<>47)&(Z<>48)&(Z<>49)AA(Z)=B
9150IF((Z<41)&(Z>23))&(Z<>334)&(Z<>35)&(Z<>36)&(Z<>28)AA(Z)=H
9160N. Z;AA62=9;AA29=9;AA60=9;AA71=C
9170$E="";S=0;T=0;M=0;$G="-"
9172IFRND>0G. 9180
9173P. $7;P. $7
9175Z=A. R. %2+57;Q=A. R. %2+6;AA(Z)=128;AA(Z-Q)=111
9180 RETURN
```

This next listing is the variation on draughts, REVERSI. In REVERSI, you follow the standard rules of draughts with a few exceptions. There are no multiple jumps, and no kings. As well, you can move backwards and forwards as you choose (as if you had a board of kings). But the major difference between REVERSI and draughts -- and the whole point of the game -- is the result of a capture. In ordinary draughts, you take your opponent's piece off the board. In REVERSI, the piece changes to become one of yours (which is why the Victorians called it REVERSI). The first player to capture (that is, reverse) 12 pieces wins.

REVERSI

```
   5P. $12
  10GOS. 9000
  15P. $7    $7
  50GOS 7000
  60M=0
1000P. '    "LAST MOVE "$G'
1012F. I=1TO2
1015IFI=1INPUT"FROM"$E
1030IFI=2INPUT"TO"$E; $G=$E
1040IF$E="A2"FF(I)=72
1110IF$E="A4"FF(I)=71
1120IF$E="A6"FF(I)=70
1130IF$E="A8"FF(I)=69
1140IF$E="B1"FF(I)=66
1150IF$E="B3"FF(I)=65
1155IF$E="B5"FF(I)=64
11601F$E="B7"FF(I)=63
1170IF$E="C2"FF(I)=59
1180IF$E="C4"FF(I)=58
1190IF$E="C6"FF(I)=57
12001F$E="C8"FF(I)=56
1210IF$E="D1"FF(I)=53
1220IF$E="D3"FF(I)=52
1230IF$E="D5"FF(I)=51
1240IF$E="D7"FF(I)=50
1250IF$E="E2"FF(I)=46
1260IF$E="E4"FF(I)=45
1270IF$E="E6"FF(I)=44
1280IF$E="E8"FF(I)=43
1290IF$E="F1"FF(I)=40
1300IF$E="F3"FF(I)=39
1310IF$E="F5"FF(I)=38
1320IF$E="F7"FF(I)=37
1330IF$E="G2"FF(I)=33
1340IF$E="G4"FF(I)=32
1350IF$E="G6"FF(I)=31
13601F$E="G8"FF(I)=30
1370IF$E="H1"FF(I)=27
1380IF$E="H3"FF(I)=26
1390IF$E="H5"FF(I)=25
1400IF$E="H7"FF(I)=24
1410N. I
1420     AA(FF(2))=AA(FF(1)); AA(FF(1))=B
14601FABS(FF(1)-FF(2))>7; M=1; AA(FF(1)+((FF(2)-FF(1))/2))=H
1470GOS. 7000
20020=0; Z=24; D=1
2010XX1=-6; XX2=-7; XX3=6; XX4=7
20211FAA(Z)=120 G. 2110
20231FAA(Z)=9 G. 2110
20241FAA(Z)=128 G. 2110
2045U=0
2050IF(AA(Z+XX(D))=H)U=7
20551F((U=7)&(AA(Z+2*XX(D))=B))0=XX(D)
2056IF0<>0G. 2130
```



```
2090IF(D<4)D=D+1;G. 2045
2110IF(Z<72)Z=Z+1;D=1;G. 2010
2120IFQ=0G. 2240
2130AA(Z+2*Q)=AA(Z);S=S+1;AA(Z+Q)=C;AA(Z)=B
2140G. 50
2240Z=72;U=0
2250IF(AA(Z)<>C)G. 2290
2255?#8016=RND
2260IF(AA(Z-13)=H)G. 2290
2262IF(AA(Z-12)=H)G. 2290
2265IF(AA(Z-14)=H)G. 2290
2270IF(AA(Z-6)=B)&(AA(Z-12)=C)U=-6
2275IFU=-6&AA(Z-18)=H Q=Z-6
2280IF(AA(Z-7)=B)&(AA(Z-14)=C)U=-7
2285IFU=-7&AA(Z-21)=H Q=Z-7
2290IF(Q=0)&(Z>23)Z=Z-1;G. 2250
2350Y=0
2360Z=24+A. R. %48;Y=Y+1;IFY>400G. 2440
2365N=A. R. %10
2370N?#8010=RND
2380IF(AA(Z)<>C)G. 2360
2385IFY>200G. 2390
2386IF(AA(Z-13)=H)G. 2360
23871F(AA(Z-14)=H)G. 2360
2388IF(AA(Z-12)=H)G. 2360
2390D=1
2410IFAA(Z+XX(D))=B Q=XX(D);G. 2460
2420IFD<4 D=D+1;G. 2410
2430G. 2360
2440IFY>400 $E="Z";G. 7000
2460AA(Z+Q)=AA(Z);AA(Z)=B;G. 50
7000P. $30;@=0
7010?#E1=0
7020H=120;C=111;B=128
7030IFM=1;T=T+1
7040M=0
7090P. "ATOM "S"     HUMAN "T'
7100P. "       #12345678#"'
7110P. "      A# "$AA72" "$AA71" "$AA70" "$AA69"#A"'
7120 P. "      B#"$AA66" "$AA65" "$AA64" "$AA63" #B"'
7130P. "      C# "$AA59" "$AA58" "$AA57" "$AA56"#C"'
7140 P. "      D#"$AA53" "$AA52" "$AA51" "$AA50" #D"'
7150P. "      E# "$AA46" "$AA45" "$AA44" "$AA43"#E"'
7160 P. "      F#"$AA40" "$AA39" "$AA38" "$AA37" #F"'
7170P. "      G# "$AA33" "$AA32" "$AA31" "$AA30"#G"'
7180 P. "      H#"$AA27" "$AA26" "$AA25" "$AA24" #H"'
7210IF$E="Z"T=12
7220IF(T=12)P. "YOU WIN";END
7230IF(S=12)P. "I WIN";END
7320RETURN
9000DIMAA86 ;Y=0;DIMXX4;XX1=-6;XX2=-7;XX3=6;XX4=7;DIML1;DIMFF2
9060H=120;C=111;B=128;Q=0
 9070 DIM G2,E2
 9110F. Z=1TO86;AA(Z)=9
 9130IF((Z<73)&(Z>55))&(Z<>67)&(Z<>68)&(Z<>61)AA(Z)=C
```

```
9140IF((Z<54)&(Z>42))&(Z<>47)&(Z<>48)&(Z<>49)AA(Z)=B
9150IF((Z<41)&(Z>23))&(Z<>334)&(Z<>35)&(Z<>36)&(Z<>28)AA(Z)=H
9160N. Z; AA62=9; AA29=9; AA60=9; AA71=C
9170$E=""; S=0; T=0; M=0; $G="-"
91721FRND>0G. 9180
9173P. $7; P. $7
9175Z=A. R. %2+57; Q=A. R. %2+6; AA(Z)=128; AA(Z-Q)=111
9180 RETURN
```

Now, a couple of simple programs to draw pretty pictures on the screen.

STARBRIGHT

```
   1CLEAR 4
   5GOS. A
  20PLOT K, X, Y
  30PLOT K, X, (192-Y)
  40PLOT K, (256-X), Y
  50PLOT K, (256-X), (192-Y)
  60G. 5
1000END
2000X=A. R. %256+1; Y=A. R. %192+1
2100K=13+A. R. %3
2200IF(K<>15)&(A. R. %4<3)K=15
2500R.
```

TANGLEWOOD

```
   5CLEAR2
  10A=A. R. %20; B=A. R. %193
  12D=10+A. R. %60; E=20+A. R. %90; Y=1+A. R. %9; X=1+A. R. %9
  20Z=13
  90F. C=A TO B; PLOTZ, C, A; PLOTZ, C, B; PLOTZ, A, C; PLOTZ, B, C
 100N.
 110A=A+Y; B=B-X
 170IFB-A>=1 G. 90
 175X=4+A. R. %4; Y=1+A. R. %12
 180IFZ=13 Z=15; A=A. R. %10; B=A. R. %60+122; G. 90
 190G. 10
1000END
```

FRANTIC is a program which graphically illustrates how quickly the Atom thinks. A fly (disguised as a hash symbol) tries to find its way past obtacles to the bottom left hand of the screen. You'll understand why it is called FRANTIC when you watch its desperate attempts to escape.

FRANTIC

```
 10CLEAR0
 12MOVE0,0
 14PLOT1,63,0;PLOT1,0,47;PLOT1,-63,0;PLOT1,0,-47
 15 GOS. 2000
 20P=A. R. %9+1;Q=P;P?#8000=#23
 22S=0
 25TQ=P
 27Z=0
 30EZ=A. R. %12+1
 40M=P+AA(Z)
 45IF(M?#8000= #40)P=M
 50IF P<>Q G. Y
 55G. E
 60Y0?#8000=#40;P?#8000=#23
 70S=S+1
 80IF(P>474)GOS. R
900G. T
1000END
2000DIMAA12
2010AA1=033;AA2=01;AA3=32;AA4=31;AA5=-1;AA6=-31;
    AA7=-32;AA8=-33
2015AA9=33;AA10=1;AA11=33;AA12=-31
2020F. J=1TO75;Z=A. R. %430+10;Z?#8000=#DE;N.
2025F. J=445TO447;J?#8000=#70;N.
2030RETURN
3000RP. $7;P. $30;P. "TIME TAKEN "S
3010F. R=1TO9;P. $7;N. ;G. 10
```

HOT SAUCE needs no explanation. It's all within the program.

HOT SAUCE

```
 5REM HOT SAUCE
10P. $12;       ?#E1=0
20DIMZ10
40IN. "WHAT'S YOUR NAME, PARDNER"$Z;P. $12;?#E1=0
90S=0;P. "OK, "$Z", I AM THINKING OF";?#E1=0
110P. "A NUMBER BETWEEN 1 AND 100"'
```

```
2440A=A+100;P. "ADVERSARY DESTROYED"';P. ';G. 2520
2480FORF=1TO8;P. ';N.
2485FORF=1TO20;WAIT;N. ;P. "TOUGH - YOU DIDN'T MAKE IT"'
2490P. $11;?#E1=0
2500FORF=1TO20;WAIT;N. ;P. "TOUGH - YOU DIDN'T MAKE IT"'
2505P. $11;?#E1=0;G. 2485
2520M=M+1;P. "MISSION:  "M';A=A+Z+1;P. '
2560P. "YOUR MARKSMAN RATING:  "A';P. '
2580P. "COULD YOU TAKE ANOTHER MISSION"'"YES(1) OR NO(2)";IN. U
2600P. $12;?#E1=0;IFU=1 G. 380
2610P. "DISMISSED"';?#E1=128;END
```

And after the horror and complexity of the last program, here's a simple and useful accounts program, based on a ZX80 program written by Ron Jones of Chandlers Ford. Money paid in must be entered as a negative number. To save the balance, balance, enter GOTO 10 after the first RUN.

```
 1B=0
 2DIMZ(1);DIMAA(6)
 5F. J=1TO6;AA(J)=0;N. J
 7M=6
10@=0
12P. $#0C
15GOS. 10000
20GOS. 900
30P. "ANY CHANGES? (Y OR N)"';INPUT$Z;IF$Z="N"G. 100
50P. "NUMBER";INPUTK;IF(K>M)\(K<1)G. 50
60P. ';P. "NEW AMOUNT"';IN. E;AA(K)=E;G. 20
92;P. F"   ";GOS. (F*1000);P. "    $"AA(F);P. '
100P. "ENTER SALARY"';IN. S;GOS. 900;R=S-T+B;P. '
126P. "BALANCE $"R';B=R
140END
900T=0;P. $#0C;P. "PERSONAL ACCOUNTS"'
910P. ';P. "PREVIOUS BALANCE "B'
915P. '
917FORF=1TO M;P. F"    "
920GOS. (F*1000)
925P. "   $"AA(F);P. '
930T=T+AA(F)
940NEXT F
950P. ';R.
1000P. "CHEQUES OUT";R.
2000P. "ACCESS";R.
3000P. "RATES";R.
4000P. "MORTGAGE";R.
5000P. "STANDING ORDERS";R.
6000P. "MONIES IN (-)";R.
10000P. "THIS IS A PERSONAL ACCOUNTS"';P. "PROGRAM.   MONIES IN"'
10010P. "MUST BE ENTERED AS A NEGATIVE"';P. "NUMBER.    TO SAVE"'
10020P. "BALANCE AFTER THE FIRST RUN"'
10025P. "ENTER GOTO 10,"'
10030P. "INSTEAD OF RUN"'
10040P. ';P. ';P. "          STAND BY"'
10050FORN=1TO10*60;WAIT;N. N;R.
```

Baccarat was first introduced into France from Italy in about 1490, during the reign of Charles VIII. It is most unlikely, historians say, that Charles played it on an Acorn Atom. The Italian game was called Baccara, and this game -- CHEMIN DE FER -- is a distant cousin of that old favourite.

You and the Atom (the banker) roll five dice each. If any die comes up 2 or 5, it must be rolled again. You add the pips on dice which did not come up 2 or 5, and then add to this the total of the pips from the dice you've rolled again. If a die comes up 2 or 5 the second time, it counts as zero. The aim is to get as close as possible to 9, or to get a two-digit number ending in 9. The program automatically strips two-digit numbers down to their final digit. Run it a few times to see how it works, and you'll begin to see why it is so popular.

CHEMIN

```
  5@=0
  7P. $#0C
  8R=0
 10B=0;P=0;G. 340
 40D=0;C=0
 60F. G=1TO5;A=ABSRND%6+1
 70F. N=1TO20;WAIT;N. N
 80IF(A=2)\(A=5)C=C+1;A=0
100P. A"  ";D=D+A;N. G;P. ';P. D" "
110IFD>9D=D-10;P. D" ";IFD>9D=D-10
180P. "TOTAL ON FIRST ROLL IS "D'
190IFD=9 P. "LA GRANDE"'
200IFD=8 P. "LA PETITE"'
210IFD=7 P. "NATURAL"'
215F. N=1TO60;WAIT;N. N
220IF(C=0)\(D=7)\(D=8)\(D=9)R.
230P. "MUST ROLL "C" AGAIN"'
235F. N=1TO60;WAIT;N. N
240F. A=1TO C;F. N=1TO60;WAIT;N. N;E=ABSRND%6+1
250IF(E=2)\(E=5)E=0
255P. E" ";D=D+E;N. A
260P. D"  ";IFD>9D=D-10;P. D" ";IFD>9D=D-10;R.
265R.
270END
340R=R+1;P. "                ROUND "R'
342F. N=1TO90;WAIT;N. N
345P. "        BANKER"';P. ';GOS. 40
347F. N=1TO90;WAIT;N. N
350P. "    FINAL TOTAL "D';J=D
```

```
390P. ´;P. "        PLAYER"´;P. ´;GOS. 40
395P. "      FINAL TOTAL "D´;P. ´
400F. N=1TO120;WAIT;N. N
420P. "BANKER      PLAYER"´
430P. "    "J"          "D´
440IFJ=D;P. "           STANDOFF"´;G. 510
460IFJ>D;P. "BANKER";B=B+1
490IFJ<D;P. "PLAYER";P=P+1
500P. " WINS"´
510P. "          TALLY AFTER ROUND "R´
515F. N=1TO60;WAIT;NEXTN
520P. "BANKER  PLAYER"´
530P. "     "B"        "P´;IFB+P=9 GOTO550
535F. N=1TO90;WAIT;N. N
537IFB+P>1;P. "             "B+P" SCORING GAMES"´
540F. N=1TO120;WAIT;N. N;P. $#0C;G. 340
550P. ´;P. "        END OF THE GAME"´;P. ´
560IFP>B P. "THE PLAYER"
570IFB>P P. "THE BANKER"
580P. " IS THE WINNER"´;P. ´;P. ´
590P. "POINTS SCORED:  "ABS(B-P)*R´
600FORN=1TO30;WAIT;N. N;G. 560
```

You can now be a master trader in this splendid game. You start your career with 200 caravans which you equip as you see fit, then you take off for foreign ports to sell your goods. If you choose safe ports, you'll get a lower return on your merchandise, and provisions will cost more. Try more hazardous zones and your profits will rise with the increases risk. You start with the princely sum of $600. The accolade KINGS RANSOM awaits the true MASTER TRADER.

MERCHANT

```
10G. 300
20IFA=1 THEN $Z="PROVISIONS"
30IFA=2 THEN $Z="LIVESTOCK"
40IFA=3 THEN $Z="COTTON"
50IFA=4 THEN $Z="JEWELS"
60RETURN
70P. "YOU NOW HAVE $"B´
80G. 680
90M=0;N=0;O=0;P=0;T=0
120RETURN
130P. "PROVISIONS:  $240"´
140P. "REPAIRS:  $40"´
150P. "CUSTOM DUTY:  $40"´
160RETURN
170P. "PROVISIONS:  $110"´
180P. "REPAIRS:  $50"´
```

```
190P. "CUSTOM DUTY:  $50"'
200RETURN
210P. "ARE YOU SURE OF YOUR"'
220P. "DESTINATION.  1-NO;  2-YES"'
230INPUTW
240P. $#0C
250RETURN
260P. "YOU ARE A MASTER TRADER OUT OF    VENICE "'
280P. "YOU MUST TRADE TO SURVIVE. "'
290RETURN
300P. $#0C
305P. "MERCHANT OF VENICE"';P. " "'
310FOR0=1TO120;WAIT;N.
315P. "WHEN YOU ARE READY PRESS "'
320P. "RETURN"'
321B=600
325LINK#FFE3
330DIMZ(10)
340@=0
350GOSUB260
360G. 510
480GOS. 230
490IFW=1 THEN END
506U=0
510P. "YOU HAVE $"B" AND 200 EMPTY       CARAVANS"'
520P. "YOUR GOODS ARE SPLIT BETWEEN: "'
525P. "PROVISIONS, LIVESTOCK, COTTON   AND JEWELS"'
530GOSUB890
540A=1
550FORC=1TO4
560GOS. 20
570A=A+1
575U=U+1
580P. "HOW MANY CARAVANS WILL YOU FILL WITH "$Z
590IF$Z="PROVISIONS"       INPUTM
591IF$Z="LIVESTOCK"       INPUTN
592IF$Z="COTTON"       INPUTO
593IF$Z="JEWELS"       INPUTP
600P. $#0C
610T=M+N+O+P
620IFT=200       G. 700
630IFT>200       G. 660
640NEXTC
650GOTO700
660P. "CHEAT.  START AGAIN. "';END
680P. "DO YOU WANT TO TRADE AGAIN"'
681P. "1 - NO;  2 - YES"
690GOTO480
691IFW=1 THEN END
700P. "WHERE WILL YOU TRADE? SPAIN(1), ENGLAND(2) OR THE "
705P. "ORIENT(3)"';P. " "'
720P. "SPAIN IS PRESENTLY AN ALLY OF     ITALY. "'
740P. "ENGLAND IS PRESENTLY NEUTRAL"'
745P. "BUT THERE IS THE THREAT OF WAR. "'
760P. "THE ORIENT IS INHABITED BY"'
765P. "HOSTILE NATIVES. "'
```

```
770P. "WHICH DESTINATION (1,2 OR 3)"
780INPUTF
790P. $#0C
800IF F=1       GOTO 840
810IF F=2       GOTO 880
820IF F=3       GOTO 920
830GOTO780
840P. "AN EASY ROUTE.  A SAFE JOURNEY"'
841P. "BUT A LOW PRICE FOR YOUR GOODS"'
842P. "PROVISIONS WILL BE EXPENSIVE"'
850GOS. 210
860IFW=1        G. 770
870G. 960
880P. "A HARD ROUTE.  YOU WILL GET A"'
881P. "GOOD PRICE FOR YOUR GOODS BUT"'
882P. "THERE IS THE THREAT OF WAR."'
883P. "PROVISIONS ARE AVERAGE PRICE"'
890GOS. 210
900IFW=1        G. 770
910G. 1050
920P. "THE HARDEST ROUTE.  YOU MUST PASS"'
930P. "THROUGH HOSTILE TERRITORY"'
931P. "YOU WILL GET A GOOD PRICE FOR"'
932P. "YOUR GOODS AND PROVISIONS"'
933P. "ARE CHEAP."'
935GOS. 210
940IFW=1        G. 770
950GOTO1290
960B=B-320
970G=ABSRND%2+14
980H=ABSRND%2+12
990J=ABSRND%2+14
1000K=ABSRND%2+16
1010P. "PROVISIONS:  $260"'
1020P. "REPAIRS:  $40"'
1030P. "CUSTOM DUTY:  $20"'
1040GOTO1520
1050B=B-280
1060G=18+ABSRND%3
1070H=16+ABSRND%2
1080J=18+ABSRND%2
1090K=18+ABSRND%3
1100L=ABSRND%9+1
1110IFL<3       G. 1150
1120IFL<6       G. 1190
1130IFL<9       G. 1210
1140IFL<11       G. 1250
1150N=0
1160P. "YOUR LIVESTOCK KILLED BY"'
1161P. "RAIDERS"'
1170GOS. 130
1180G. 1530
1190P. "SAFE ARRIVAL IN ENGLAND"'
1200G. 1270
1210GOS. 90
1220P. "ATTACKED BY RAIDERS.  ALL YOUR"'
1221P. "GOODS WERE STOLEN."'
1230GOS. 130
1240G. 1560
```

```
1250P=0
1260P. "JEWELS IMPOUNDED BY CUSTOMS"'
1261P. "NO PROFIT ON THIS JOURNEY. "'
1270GOS. 130
1280G. 1520
1290B=B-210
1300G=22+ABSRND%2
1310H=18+ABSRND%2
1320J=20+ABSRND%3
1330K=22+ABSRND%3
1340L=ABSRND%9+1
1350IFL<3 THEN G. 1390
1360IFL<6 THEN G. 1420
1370IFL<9 THEN G. 1450
1380IFL<11 THEN G. 1480
1390P. "YOUR CARAVAN WAS ATTACKED BY A"'
1391P. "MONGOL HORDE. "'
1392P. "YOU WERE TOTALLY DESTROYED. "'
1400END
1420P. "SAFE ARRIVAL AT SAMERKAND. "'
1430G. 1510
1440GOS. 90
1450P. "YOU WERE ATTACKED BY MONGOL"'
1451P. "RAIDERS.  ALL YOUR GOODS STOLEN. "'
1460GOS. 170
1470G. 1560
1480G=G/2
1490H=H/2
1500P. "CARAVAN HIT BY PLAGUE.  HALF"'
1501P. "YOUR LIVESTOCK AND PROVISIONS"'
1502P. "PERISHED. "'
1510GOS. 170
1520P. "YOUR LIVESTOCK SOLD FOR $"H*N/13'
```

```
1530P. "YOUR JEWELS SOLD FOR $"J*P/13'
1540P. "YOUR COTTON SOLD FOR $"K*O/13'
1550P. "YOUR PROVISIONS SOLD FOR $"G*M/13'
1560B=(H*N/13)+(K*P/13)+(J*O/13)+(G*M/13)+B
1570IF B<0 THEN G. 1610
1580IFB>1000 THEN G. 1650
1590GOS. 70
1600G. 680
1610P. "  "'
1620DOP. "PAUPER"'"PAUPER"'; U. O
1650DOP. "YOU HAVE $"B" MASTER TRADER"'
1660P. "KINGS RANSOM"'; U. O
```

This program puts you in control of a space ship made up of a less than, an equals and a greater than sign. You are at the top of the screen and a horde of angry alien ships (arrows) are heading towards you. You start with a score of two million and every time you hit a ship you lose points. SHIFT and REPEAT move you left and right. You score dramatically decrements if an alien hits your equal sign.

HARLEM

```
  5S=2000000; T=0
 10P. $12; ?#E1=0
 15F. J=1T015; P. ' '; N.
 20X=1
 40F. U=1T0500
 50 A=A. R. %32+449; A?#8000=#1E
 60 A=A. R. %32+449; A?#8000=#1E
 70IF(?#B001&#80=0)&(X>1) X=X-2
 80IF(?#B002&#40=0)&(X<28 )X=X+2
 85M=X+33; N=X+1; Q=N+1
 87X?#8000=#3C; N?#8000=#3D; Q?#8000=#3E
 90IFM?#8000<>32 GOS. P
100P. ' '; ?#E1=0
110N.
120P. "YOUR SCORE WAS "S'
130IF S<T G. 140
135T=S
140P. "HIGHEST SCORE SO FAR"T'
150P. "PRESS ANY KEY TO PLAY AGAIN"'
160LINK#FFE3
170G. 10
1000END
1500PF. J=1TO 9; M?#8000=128; M?#8000=#23; S=S-1777
1550?#B002=?#B002: 4
1600P. $30"SCORE IS "S'; N. ; M?#8000=32
1650F. J=1T015; P. ' '; N. ; R.
```

Having survived the angry aliens, you're ready to try landing on the moon.

MOONLANDER

```
  1P. $30; @=0; CLEAR0; Q=ABSRND%9*10+200; P. Q'
 10FORF=0T031; F?#81E0=127; N.
 20FORF=11060; WAIT; N.
 25X=ABSRND%25; X?#81C8=#FF
 30A=ABSRND%20+7; A?#8000=30
100IF?#B001=127 B=1
```

```
110IF?#B001=191 B=-1
120IF?#B001=247 B=-64
1300=Q-10;IFQ<0 G.300
135P.$11;?#E1=0;P.Q"   "'
140A=A+B;FORF=1TO20;WAIT;N.;A?#8000=32;A=A+32;A?#8000=30
142E=A-B-32;E?#8000=32
145D=A+32;IFD?#8000=127 G.300
146IFD?#8000=#FF G.400
150G.100
300P.$7;A?#8000=158;FORF=1TO20;WAIT;N.
310A?#8000=#66;FORF=1TO20;WAIT;N.
320A?#8000=#41;FORF=1TO20;WAIT;N.
330A?#8000=32;END
400FORF=1TO10;P.' ';N.
410P."<SAFE DOCKING>"';P.$11;?#E1=0
420FORF=1TO20;WAIT;N.
430P."<SAFE DOCKING>"';P.$11;?#E1=0
440FORF=1TO20;WAIT;N.;G.410
```

The next three programs are reprinted from INTERFACE, the monthly magazine printed by the Independent Atom Users' Group.

The first one is another type of moonlanding program. In this, we are told, Captain Atomic, in the good ship ACORN, is descending to the surface of the moon, He has to set the controls to autopilot, and it is your job as the computer controlled landing system to set him down as gently as possible - preferably without destroying the ship.

```
10  DIM RR(4), P(-1): L=#B002: ?#E1 = 0: P.$21
20  [
30  :RR0 LDA L: LDY#81
40  :RR1 LDX #80
50  :RR2 DEX: BNE RR2
60  EOR @ 4: STA L : DEY : BNE RR1
70  RTS: ] : P.$6: G.100
80  ?#81 = 255
90  FOR N = X TO Y STEP Z: ?#80 = N: LINK RR0: N.:R
100 P.$12: P. "LUNAR LANDER" ' ;P.'
110 P. "TRY TO LAND YOUR SPACESHIP ONTO" '
120 P. "THE SURFACE OF THE MOON."'
130 P. "THE AMOUNT OF THRUST YOU USE" '
140 P. "DETERMINES YOUR RATE OF DESCENT" ';P.'
150 P. "PRESS RETURN TO START YOUR (10 spaces)
    DECENT" '
160 LINK#FFE3: @ = 0: GOS.360
170 P.$12: P. "HEIGHT * * * FUEL * * * VELOCITY" '
180 IF H<. 1000 P."0"
190 IF H<100 P."0"
200 P.H" (5 spaces)"F"(7 spaces)"S'
210 FOR A = 1 TO 8-H/500: P.':N
220 FOR B = 1 TO 11*(ABSRND%4 + 1): P. "_";N.
230 P. "<*>" '
```

```
240  FOR A = 8 - H/500 TO 8; P.';N.
250  P. "(32 inverse Xs) " '
260  IN. "THRUST" T; IF F - T<1 T = 1
270  S = S + S/10 + 5 - T; H = H - 5; IF H>1600 G.340
280  F = F - T/3; IF H>0 G. 170
290  IF S>10 G.320
300  X = 60; Y = 10; Z = -10; FOR Q = 1 TO 3; GOS.80; N.
310  P. "SUCCESSFUL LANDING: "F*23" POINTS" '; END
320  X = 1; Y = 100; Z = 3; GOS.80
330  P. "CRASH LANDING AT "S" KM/HR " '; G.330
340  P.$12; X = 100; Y = 1; Z = -2; GOS. 80
350  P. "ESCAPE VELOCITY <*> " '; G. 340
360  H = 1500 + ABSRND % 99 + 1; F = 90 + ABSRBD % 74 + 1
370  S = 10 + ABSRND % 9 + 1; R.
```

Under your command is the minesweeper Sembler, and you are in pursuit of a pack of submarines. To set up your Atom Sonar Unit you have to choose the area of your search sector (between 5 and 8 Km is best). You must then set the co-ordinates for the detonation of the first series of depth charges. The sonar report tells you how close you were. Can you home in and destroy the subs before they get you?

```
10   DIM RR(4), P(-1); L =# B002; ?#E1 = 0; P.$21
20   [
30:  RR0 LDA L; LDY #81
40:  RR1 LDX#80
50:  RR2 DEX; BNE RR2
60   EOR @ 4; STA L; DEY; BNE RR1
70   RTS; ]; P.$6; G.100
80   ?#81 = 255
90   FOR N = J TO K STEP M; ?#80 = N; LINK RR0; N.;
     R.
100  P.$12; P. "SUB STRIKE" '; P.'
110  @ = 0; IN. "AREA OF SEARCH SECTOR" A ; P.'
120  B = (A/3) + 3; C = ABSRND% A + 1
130  D = ABSRND % A + 1; E = ABSRND% A + 1
140  FOR F = 1 TO B; P.';P. "SWEEP" F '
150  IN. "N - S CO-ORDINATE" Y; IN. "E-W
     CO-ORDINATE" X
160  IN. "DEPTH" Z ;P.';P.'
170  IF ABS (X-C) + ABS (Y-D) + ABS (Z-E) = 0 G.230
180  GOS. ;P.';N.
190  P. 'ENEMY SUB DESTROYED YOU" '
200  J = 10; K = 80; M = 3; GOS.80
210  P. "SUB WAS AT "C"    " D "  " E
220  FOR F = 1 TO 60; WAIT; N.; G.210
230  P "YOU DESTROYED THE SUB IN "F" TRIES" '
240  J = 100;K = 2; M = -2;GOS.80
250  P. "WARNING "" ANOTHER SUB" '
260  FOR Q = 1 TO240; WAIT; N.; P.$12; G.120
270  J = 10; K = 300; M = 50; GOS.80
280  P.$12; P. "SONAR SCAN REPORT:" ' "SHOT WAS"
290  IF Y>D P. "NORTH"
300  IF Y<D P. "SOUTH"
310  IF X>C P. "EAST"
320  IF X<CP. "WEST"
330  P.";"
340  IF Z>E P. "TOO DEEP" '
350  IF Z<E P. "TOO SHALLOW" '
360  IF Z = E P. "DEPTH OK" '
370  P. '; P.'; P. "PRESS RETURN TO CONTINUE SCAN";
     LINK# FFE3; P$ 12; R.
```

The Atom version of the classic game 'Rock, Scissors, Paper'. Try to think ahead of the computer and guess its choice, but you'll find it difficult. The Atom seems blessed with psychic qualities.

```
10   P.$12; FORF = 1 TO 50; WAIT; N.
20   P."OUTSMART" '; FORF = 1 TO 60; WAIT; N.
30   P. "HOW MANY GOES DO YOU WANT"; IN.Y
40   D = 0; G.100
50   C = ABSRND%2+1
60   IF C = 1 P. "PAPER" '
70   IF C = 2 P. "SCISSORS" '
80   IF C = 3 P. "ROCK" '
90   R.
100  A = 1; FORT = 1 TO Y; P. "TRY NUMBER "
110  FORF = 1 TO A; P.$7; N.; P.A'
120  P. "1 = PAPER " ' " 2 = SCISSORS " ' " 3 = ROCK  " ';P.
130  P. "WHAT IS YOUR CHOICE"; IN.B
```

```
140  IF B< 1 OR B>3 G.130
150  IF B = 1 P. "PAPER" '
160  IF B = 2 P. "SCISSORS" '
170  IF B = 3 P. "ROCK" '
180  P. "MY CHOICE IS ...."; GOS.50
190  IF C = B G.280
200  IF C>B G.250
210  IF C = 1 G.230
220  P' "OK -- YOU WIN" '; D = D + 1; G.290
230  IF B<>3 G.220
240  P' "I WON THIS TIME" .; P.'; G.290
250  IF B<>1 G.240
260  IF C<>3 G.310
270  G.220
280  P. "WE'VE DRAWN" '; P.'
290  IF T = Y G.320
300  P. "PRESS RETURN FOR YOUR NEXT GO" '; LINK
     FFE3; P.$12
310  A = A + 1; N.
320  FOR F = 1 TO 120; WAIT; N.
330  P.$12; P. "THAT'S ALL ..." '
340  W = A - D - 1; IF W<0W = 0
350  P. "I WON " W" GAMES "' " YOU WON "D" GAMES"
     '; END
```

Finally, here are a number of programs, some of which have been introduced, in piece-meal fashion, in the text earlier. Some of them are slightly different from the listings given when the programs were being explained as with some programs we kept on working on them before dumping them from the printer.

DODGEM

```
  1T=0
  5S=234567
  6P. $12; ?#E1=0
  7K=1; A=12
 15F. J=1TO15; P. ' '; N.
 20X=7; A=7
 40F. U=1TO232
 45IF(A< 9)\(A>18)K=-K
 50A=A+A. R. %2*K; P=A+8; A?#81C0=#3C; P?#81C0=#3E
 52R=R. %27
 55IFR=9; Q=A+1+A. R. %6; Q?#81C0=#4A
 56IFR=4; Q=A+2; Y=A+3; E=A+4; Q?#81C0=#30; Y?#81C0=#09; E?#81C0=#0C
 58IFR=1; Q=A+3; Y=A+4; E=A+5; Q?#81C0=#89; Y?#81C0=#83; E?#81C0=#85
 70IF(?#B001&#80=0)&(X>1) X=X-1
 80IF(?#B002&#40=0)&(X<30 )X=X+1
 85M=X+32
 87X?#8000=#08
 90IFM?#8000<>32 GOS. P
100P. ' '; ?#E1=0
110N.
120P. "YOUR SCORE WAS "S'
130IF S<T G. 140
135T=S
140P. "HIGHEST SCORE SO FAR"T'
145F. J=1TO3; P. $7; N.
```

```
 150P. "PRESS ANY KEY TO PLAY AGAIN"'
 160LINK#FFE3
 170G. 5
1000END
1400PW=?(#8000+M)+128: Z=W-128
1500F. J=1TO11: X?#8000=#88: M?#8000=W        : X?#8000=#08: S=S-77
1550M?#8000=Z

1600P. $30"POINTS"'S ; N. ; M?#8000=32
1650F. J=1TO15: P. ' ; N. ; R.
```

MOULIN

```
   1CLEAR0: U=0: ?#E1=0: K=4
  10DIMV(5): !V=#04081020: V!4=#102
  20A=A. R. %34+20: B=A. R. %30+8
  25G. T
  30F. J=1TO10: PLOT13, (A. R. %64), (A. R. %48): N.
  32TX=A: Y=B+K/4
  33P. $30; P. U
  35PLOT13, A, B: GOS. P
  40IFQ=1  P. $7: END
  50M=A: N=B
  55U=U+177
  60IF(?#B001&#80=0)&(A>2)A=A-2
  70IF(?#B002&#40=0)&(A<62)A=A+2
  72PLOT13, (A. R. %64), (A. R. %48)
  80B=B+K: IF(B<4)\(B>42)K=-K
  90PLOT15, M, N: G. T
 120PP=X/2+(47-Y)/3*32+#8000
 130Q=(?P&(V?(X&1+(47-Y)%3*2))<>0): RETURN
1000END
```

PURSUIT

```
   1P. $21
  10DIMVV(4), P(-1): L=#B002: [: VV0 LDA L: LDY #81: : VV1 LDX #80
  15: VV2 DEX: BNE VV2: EOR @4: STA L: DEY: BNE VV1: RTS: ]
  17G. 90
  90P. $6: G. B
 100AIF?#B001&#80=0 A=A-3*Y
 120IF?#B002&#40=0 B=B-3*X
 140A=A+Y: B=B+X
 145?#80=3: LINK VV0
 150F=F+P: G=G+Q
 155IFR. %9>7 F=F+R. %4
 160IF(A<1)\(A>127)Y=-Y
 170IF(F<1)\(F>127)P=-P
 180IF(B<1)\(B>63)X=-X
 190IF(G<1)\(G>63)Q=-Q
 200PLOT15, C, D: C=A: D=B: PLOT13, A, B
 210PLOT15, H, J: H=F: J=G: PLOT13, F, G
 220IF ABS(A-F)+ ABS(B-G) <3 G. C
```

```
 5006. A
1000BA=1; B=1; Y=2; X=2
1005CLEAR1
1010      F=A. R. %128; G=A. R. %60; P=1+A. R. %2; Q=1+A. R. %2
1020G. A
2000CMOVEA, B
2010PLOT5, A, B
2020A=A+R. %3
2030B=B+R. %3
2035?#80=A. R. %30+8; LINK VV0
2040G. 2010
2050END
```

COLLECTOR

```
  10G. 5000
  20AIF?#B001&#80=0 A=A-2*Y; T=T+10
  30IF(?#B002&#40=0)B=B-2*X; T=T+10
  50A=A+Y; B=B+X; IF(A<1)\(A>127)Y=-Y
  51IF(B<1)\(B>63)X=-X
  52PLOT15, C, D
  55C=A; D=B
  60PLOT13, A, B
  70IFT=2000END
  80T=T+1; ?#8000=T
 9906. A
1000END
5000A=64; B=32
5002Y=1; X=1
5005CLEAR1
5030F. J=1TO50
5040PLOT13, (A. R. %128), (A. R. %64); N.
5050T=0
5900G. A
```

IMPLODERS

```
  5P. $12; P. $21
 10DIMVV(4), P(-1); L=#B002
 30[; .VV0 LDA L; LDY #81; .VV1 LDX #80; .VV2 DEX; BNE VV2
 33; N.
 80EOR @4; STA L; DEY; BNE VV1; RTS; ]
 82Y=1; P=449
 85P. $6; N=0; B=0
 87G. 210
100J?#81=255
110?#80=ABSRND%3+2; LINK VV0
200RETURN
210CLEAR0
211I=0
215P.
220F. A=1TO128; P. $#EE+R. %13; N.
```

```
230KP?#8000=#C7
235P=P+Y; T=P-Y
240IF(P<450)\(P>478)Y=-Y
245X=2*(P-448-Y)+1; PLOT13, X, 6
247W=6
250IF?#B001%#80=0 GOS. 0
260G. E
5000PLOT13, X, W; N=N+3
501IFRND%12>10G. Z
505IF((W>8)%(W<46))G. U
506IFW=48 B=B+1673
510?#80=23+RND%9; LINK VV0
512IFW>45G. Z
515UPLOT15, X, W
520IFW<48W=W+1; G. 0
540ZR.
880ET?#8000=32
890I=I+1
895IF I=600 P. $7; P. "SCORE "B-N; END
900GOS. J; G. K
1000END
```

SWAPPO

```
10P. $12; P. "SWAPPO"'; P. '; @=0; ?#E1=0
15DIMAA(20); X=1
20B=9; AA(1)=ABSRND%9+1
30FORC=2TO B
40AA(C)=ABSRND%9+1; X=1
45FOR D=1 TO C-1
50IFAA(C)=AA(D) X=3
55N. D
56IFX=3 G. 40
60X=1; N.
70P. "PRESS RETURN TO FIND OUT YOUR"'"STARTING SEQUENCE"'
75LINK#FFE3; E=0; P. '; P. $12; GOS. 200
80P. '; P. '; P. "HOW MANY TO SWAP"; IN. F
90IFF=0 G. 170
100IFF<=B G. 120
110P. "I CAN ONLY SWAP "B'; G. 80
120E=E+1; FORC=1TO F/2
130G=AA(C); AA(C)=AA(F-C+1); AA(F-C+1)=G
140N. ; P. $30; @=0; GOS. 200
150C=1
151IFAA(C)<>C G. 80
152C=C+1; IFC<>9 G. 151
160P. "YOU SUCCEEDED IN "E" MOVES"'; P. '
170P. "DO YOU WANT TO TRY AGAIN."'"YES(1), NO(2)"'
180IN. N; IFN=1 G. 10
190P. "THANKS FOR THE GAME"'; ?#E1=128; G. 190
200?#E1=0
210FORC=1TO B; P. AA(C)" "; N. ; P. '; P. '
212?#8091=32
215FORY=1TO10
220FORW=0TO15; W?#8080=W?#8080+128; N.
222FORW=0TO15; W?#8080=W?#8080-128; N. ; N.
230R.
```

MALACHI

```
 10P. $12; @=0; C=0; D=150
 50P. "        ***********"'
 60F. E=1TO4; P. "        "$#2A" "$128" "$128" "$128" "$128, $#2A'
 70P. "        "$#2A, $128" "$128" "$128" "$128" "$#2A'; N.
 80P. "        ***********"'; P. '
110F=71; FORG=1TO4; F?#8000=#A1; F=F+64; N.
160H=174; H?#8000=#AA
180P. "WHICH DIRECTION WILL YOU MOVE"; IN. A; P $11
200C=C+1; F=71+((ABSRND%4-1)*64)
250K=ABSRND%4+1; IFK>2 G. 310
270IFK=1 F=F+33
280IFK=2 F=F-31
290IFF?#8000=74 G. 320
```

```
300F?#8000=#BE
310H?#8000=127
320IFA=1 H=H-33
330IFA=2 H=H+31
340IFA=3 H=H+33
350IFA=4 H=H-31
360IFH?#8000=74 G. 160
370H?#8000=#AA; IFC<5 G. 180
400IF?#8028<>190; IF?#8068<>190; IF?#80A8<>190; IF?#80E8<>190 G. A
410G. 180
415AT=71; R=0
420IFT?#8000=#A1 C=C+10
425T=T+64; R=R+1; IFR<4 G. 420
430FORW=1TO60; WAIT; N. ; P. '
440P. "END OF GAME"''D-C" POINTS"; END
```

TARTAN ARMY

```
  5P. $12;@=0
 10FORX=1TO320;P. $128;N.
 20T=0;FORF=0TO319;K=ABSRND%25;IFK=1 F?#8000=#8F
 30IFK<>1 F?#8000=127
 40N.
120X=ABSRND%320;X?#8000=#AA
130M=0
150FORF=1TO320;IFF?#8000=#8F M=M+1
160N.
240P.'"HOOLIGANS ARRESTED: "T"  "'"POLICE VANS LEFT: "M"  "'
250P."       "'';P. $11;P. $11
260IFM<1 G. C
370IN. Z;W=X+Z
390IFW?#8000=143 G. A
400IFW?#8000=190 G. B
405IFW?#8000<>127 P. $11;G. 370
406X?#8000=127
410W?#8000=170;X=W
440FORF=1TOABSRND%9+1;Q=ABSRND%320;IFQ?#8000=170 G. C
450Q?#8000=#BE;N.
470FORF=1TO4;P. $11;N.;G. 130
500AT=T-5;G. 406
510BT=T+1;G. 406
520CP. $7;FORF=1TO100;X?#8000=#66;WAIT;X?#8000=#59;WAIT;N.
530X?#8000=#BA;FORF=1TO4;P. $11;N.;P.''''';FORF=1TO4;P. $11;N.
540P."YOU HAVE BEEN HIT"';P. $11
550P."YOU HAVE HAD IT"';P. $11;G. 540
```

REFERENCE SECTION - APPENDIX

All statement and function names in the Atom BASIC can be abbreviated to the shortest sequence of letters needed to distinguish the name followed by a full stop. This can make listings look a bit obscure, but the saving in memory more than compensates for unreadable programs. If you use the abbreviations long enough, you will find that they start to come naturally.

The abbreviations are:

NAME:	ABBREVIATION:
ABS	A.
AND	A.
BGET	B.
BPUT	B.
CH	
CLEAR	
COUNT	C.
DIM	
DO	
DRAW	
END	E.

EXT	E.
FIN	F.
FOR	F.
FOUT	FO.
GET	G.
GOSUB	GOS.
GOTO	G.
IF	
INPUT	IN.
LEN	L.
LET	L.
LINK	LI.
LIST	L.
LOAD	LO.
MOVE	
NEW	N.
NEXT	N.
OLD	
OR	
PLOT	
PRINT	P.
PTR	
PUT	
REM	
RETURN	R.
RND	R.
RUN	
SAVE	SA.
SGET	S.
SHUT	SH.
SPUT	SP.
STEP	S.
THEN	T.
TO	
TOP	T.
UNTIL	U.
WAIT	

You will probably never use some of these abbreviations, but there are some you will use all the time. P. for PRINT, R. for RETURN and N. for NEXT are by far the most common three.

As well as using the ASCII code for characters, the Atom has its own code for POKEing characters onto the VDU. This code is mentioned in section 18.5 of the manual, but nowhere is there a table listing the characters and their corresponding codes. The following table charts the codes of the major part of the character set:

CODE (decimal)	HEX	CHARACTER
0	00	@
1	01	A
2	02	B
3	03	C
4	04	D
5	05	E
6	06	F
7	07	G
8	08	H
9	09	I
10	0A	J
11	0B	K
12	0C	L
13	0D	M
14	0E	N
15	0F	O
16	10	P
17	11	Q
18	12	R
19	13	S
20	14	T
21	15	U
22	16	V
23	17	W
24	18	X
25	19	Y
26	1A	Z
27	1B	[
28	1C	\
29	1D	]
30	1E	↑
31	1F	←
32	20	
33	21	!
34	22	"
35	23	#
36	24	$
37	25	%
38	26	&
39	27	'
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	-
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8

57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?

Characters 64 - 127 are the white graphic symbols

128	80	inv @
129	81	inv A
130	82	inv B
131	83	inv C
132	84	inv D
133	85	inv E
134	86	inv F
135	87	inv G
136	88	inv H
137	89	inv I
138	8A	inv J
139	8B	inv K
140	8C	inv L
141	8D	inv M
142	8E	inv N
143	8F	inv O
144	90	inv P
145	91	inv Q
146	92	inv R
147	93	inv S
148	94	inv T
149	95	inv U
150	96	inv V
151	97	inv W
152	98	inv X
153	99	inv Y
154	9A	inv Z
155	9B	inv [
156	9C	inv \
157	9D	inv]
158	9E	inv ↑
159	9F	inv ←
160	A0	inv space
161	A1	inv !
162	A2	inv "
163	A3	inv #
164	A4	inv $
165	A5	inv %
166	A6	inv &
167	A7	inv '
168	A8	inv (
169	A9	inv)
170	AA	inv *
171	AB	inv +
172	AC	inv ,
173	AD	inv -
174	AE	inv .

175	AF	inv /
176	B0	inv 0
177	B1	inv 1
178	B2	inv 2
179	B3	inv 3
180	B4	inv 4
181	B5	inv 5
182	B6	inv 6
183	B7	inv 7
184	B8	inv 8
185	B9	inv 9
186	BA	inv :
187	BB	inv ;
188	BC	inv <
189	BD	inv =
190	BE	inv >
191	BF	inv ?

Characters 192 - 255 are the grey graphic symbols

onthly magazine INTERFACE, published by the Independent Acorn Atom Users' Group ssociation with the National ZX80 and ZX81 Users Club), is just £1 for a sample £8.50 a year. Each issue has programs, tips, letters and contact addresses.

USEFUL ADDRESSES:

Acorn Computer Ltd., 4a Market Hill, CAMBRIDGE, CB2 3NJ. Acornsoft is at the same address.

- *Independent Atom Users' Group, 44 - 46 Earls Court Road, London, W8 6EJ*

Atom User Group, 'Sheerwater', Yealm View Road, Newton Ferrers, South Devon. (R. G. Meredith)

Mine of Information, 1 Francis Avenue, St Albans, Herts, AL3 6BL. Microcomputer consultancy and booksellers. Their regularly updated catalogue contains around 100 books on micros and the like.

Manchester Atom Users' Group, Clement M Rutter, 3 Leopold Ave., Withington, Manchester, M20 8JG

Computers for All, 72 North Street, Romford. (Romford) 60725 Atoms, software, books.

Timedata, 57 Swallowdale, Basildon, Essex. Publish THE ATOM MAGIC BOOK

code prijs
42,-
Vogelzang

Another great book from
INTERFACE